



# JSF

**tutorialspoint**

SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

Java Server Faces (JSF) is a Java-based web application framework intended to simplify development integration of web-based user interfaces. JavaServer Faces is a standardized display technology, which was formalized in a specification through the Java Community Process.

This tutorial will teach you basic JSF concepts and will also take you through various advance concepts related to JSF framework.

## Audience

---

This tutorial has been prepared for the beginners to help them understand basic JSF programming. After completing this tutorial, you will find yourself at a moderate level of expertise in JSF programming from where you can take yourself to the next levels.

## Prerequisites

---

Before proceeding with this tutorial you should have a basic understanding of Java programming language, text editor, and execution of programs etc. Since we are going to develop web-based applications using JSF, it will be good if you have an understanding of other web technologies such as HTML, CSS, AJAX, etc.

## Copyright & Disclaimer

---

© Copyright 2017 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

## Table of Contents

---

About the Tutorial.....	i
Audience .....	i
Prerequisites .....	i
Copyright & Disclaimer.....	i
Table of Contents .....	ii
 1. JSF – OVERVIEW .....	 1
What is JSF? .....	1
Benefits.....	1
JSF UI Component Model .....	1
 2. JSF – ENVIRONMENTAL SETUP .....	 2
System Requirement.....	2
Environment Setup for JSF Application Development .....	2
 3. JSF – ARCHITECTURE .....	 10
What is MVC Design Pattern?.....	10
JSF Architecture .....	10
 4. JSF – LIFE CYCLE.....	 12
Phase 1: Restore view .....	12
Phase 2: Apply request values.....	12
Phase 3: Process validation .....	13
Phase 4: Update model values .....	13
Phase 5: Invoke application.....	13
Phase 6: Render response .....	13

5.	JSF – FIRST APPLICATION .....	14
	Create Project .....	14
	Add JSF Capability to Project .....	15
	Prepare Eclipse Project .....	17
	Import Project in Eclipse .....	19
	Configure Faces Servlet in web.xml .....	19
	Create a Managed Bean .....	21
	Create a JSF page .....	21
	Build the Project .....	22
	Deploy WAR file .....	23
	Run Application .....	23
6.	JSF – MANAGED BEANS .....	24
	@ManagedBean Annotation .....	25
	Scope Annotations .....	25
	@ManagedProperty Annotation .....	25
7.	JSF – PAGE NAVIGATION .....	29
	Implicit Navigation .....	29
	Auto Navigation in JSF Page .....	29
	Auto Navigation in Managed Bean .....	30
	Conditional Navigation .....	31
	Resolving Navigation Based on from-action .....	33
	Forward vs Redirect .....	34
8.	JSF – BASIC TAGS .....	44
	h:inputText .....	46
	h:inputSecret .....	52
	h:inputTextarea .....	57

h:inputHidden .....	63
h:selectBooleanCheckbox .....	69
h:selectManyCheckbox .....	76
h:selectOneRadio .....	85
h:selectOneListbox .....	94
h:selectManyListbox .....	103
h:selectOneMenu .....	112
h:outputText .....	121
h:outputFormat .....	123
h:graphicImage .....	126
h:outputStylesheet .....	130
h:outputScript .....	135
h:commandButton .....	140
h:Link .....	145
h:commandLink .....	151
h:outputLink .....	157
h:panelGrid .....	163
h:message .....	170
h:messages .....	174
f:param .....	179
f:attribute .....	183
h:setPropertyActionListener .....	187
<b>9. JSF – FACELET TAGS.....</b>	<b>191</b>
Template Tags .....	192
Creating Template .....	193
ui:param Tag .....	200
Parameter to Section of a Template .....	201

Parameter to Template .....	201
Custom Tag .....	205
ui:remove Tag .....	210
10. JSF - CONVERTOR TAGS .....	214
f:convertNumber .....	214
f:convertDateTime .....	218
Custom Converter .....	223
11. JSF - VALIDATOR TAGS .....	231
f:validateLength .....	232
f:validateLongRange .....	236
f:validateDoubleRange .....	239
f:validateRegex .....	243
Custom Validator .....	247
12. JSF – DATATABLE .....	254
Display DataTable .....	255
Add Data to DataTable .....	266
Edit Data of a DataTable .....	270
Delete Data of a DataTable .....	273
Using DataModel in a DataTable .....	276
13. JSF – COMPOSITE COMPONENTS .....	280
Define Custom Component .....	280
Use Custom Component .....	282
14. JSF – AJAX .....	287

15. JSF – EVENT HANDLING .....	292
<b>valueChangeListener</b> .....	<b>292</b>
<b>actionListener</b> .....	<b>300</b>
<b>Application Events</b> .....	<b>305</b>
16. JSF - JDBC INTEGRATION.....	312
17. JSF - SPRING INTEGRATION.....	321
18. JSF - EXPRESSION LANGUAGE .....	331
19. JSF - INTERNATIONALIZATION .....	334

# 1. JSF – Overview

## What is JSF?

---

**JavaServer Faces** (JSF) is a MVC web framework that simplifies the construction of User Interfaces (UI) for server-based applications using reusable UI components in a page. JSF provides a facility to connect UI widgets with data sources and to server-side event handlers. The JSF specification defines a set of standard UI components and provides an Application Programming Interface (API) for developing components. JSF enables the reuse and extension of the existing standard UI components.

## Benefits

---

JSF reduces the effort in creating and maintaining applications, which will run on a Java application server and will render application UI on to a target client. JSF facilitates Web application development by -

- Providing reusable UI components
- Making easy data transfer between UI components
- Managing UI state across multiple server requests
- Enabling implementation of custom components
- Wiring client-side event to server-side application code

## JSF UI Component Model

---

JSF provides the developers with the capability to create Web application from collections of UI components that can render themselves in different ways for multiple client types (for example - HTML browser, wireless, or WAP device).

JSF provides -

- Core library
- A set of base UI components - standard HTML input elements
- Extension of the base UI components to create additional UI component libraries or to extend existing components
- Multiple rendering capabilities that enable JSF UI components to render themselves differently depending on the client types



## 2. JSF – Environmental Setup

This chapter will guide you on how to prepare a development environment to start your work with JSF Framework. You will learn how to setup JDK, Eclipse, Maven, and Tomcat on your machine before you set up JSF Framework.

### System Requirement

---

JSF requires JDK 1.5 or higher so the very first requirement is to have JDK installed on your machine.

<b>JDK</b>	1.5 or above
<b>Memory</b>	No minimum requirement
<b>Disk Space</b>	No minimum requirement
<b>Operating System</b>	No minimum requirement

### Environment Setup for JSF Application Development

---

Follow the given steps to setup your environment to start with JSF application development.

#### Step 1: Verify Java installation on your machine.

Open console and execute the following **Java** command.

OS	Task	Command
<b>Windows</b>	Open Command Console	c:\> java -version
<b>Linux</b>	Open Command Terminal	\$ java -version
<b>Mac</b>	Open Terminal	machine:~ joseph\$ java -version

Let's verify the output for all the operating systems:

OS	Generated Output
<b>Windows</b>	java version "1.6.0_21"  Java(TM) SE Runtime Environment (build 1.6.0_21-b07)  Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
<b>Linux</b>	java version "1.6.0_21"  Java(TM) SE Runtime Environment (build 1.6.0_21-b07)  Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
<b>Mac</b>	java version "1.6.0_21"  Java(TM) SE Runtime Environment (build 1.6.0_21-b07)  Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)

## Step 2: Set Up Java Development Kit (JDK).

If you do not have Java installed then you can install the Java Software Development Kit (SDK) from Oracle's Java site: [Java SE Downloads](#). You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally, set PATH and JAVA\_HOME environment variables to refer to the directory that contains java and javac, typically java\_install\_dir/bin and java\_install\_dir respectively.

Set the **JAVA\_HOME** environment variable to point to the base directory location where Java is installed on your machine.

For example -

OS	Output
<b>Windows</b>	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21
<b>Linux</b>	Export JAVA_HOME=/usr/local/java-current
<b>Mac</b>	Export JAVA_HOME=/Library/Java/Home

Append Java compiler location to System Path.

OS	Output
<b>Windows</b>	Append the string ;%JAVA_HOME%\bin to the end of the system variable, Path.
<b>Linux</b>	Export PATH=\$PATH:\$JAVA_HOME/bin/
<b>Mac</b>	Not required

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java. Otherwise, carry out a proper setup according to the given document of the IDE.

### Step 3: Set Up Eclipse IDE.

All the examples in this tutorial have been written using Eclipse IDE. Hence, we suggest you should have the latest version of Eclipse installed on your machine based on your operating system.

To install Eclipse IDE, download the latest Eclipse binaries with WTP support from <http://www.eclipse.org/downloads/>. Once you download the installation, unpack the binary distribution into a convenient location. For example, in C:\eclipse on Windows, or /usr/local/eclipse on Linux/Unix and finally set PATH variable appropriately.

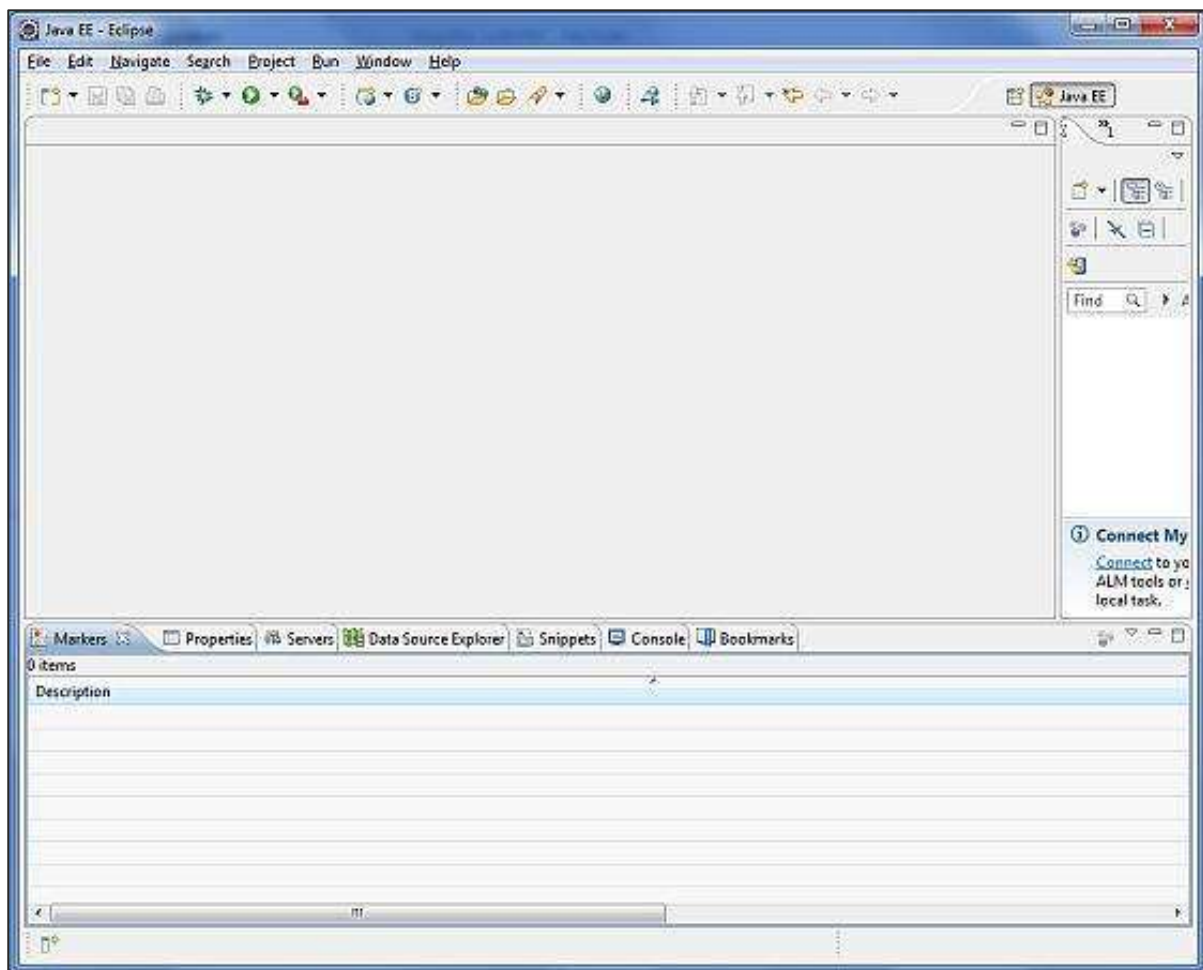
Eclipse can be started by executing the following commands on Windows machine, or you can simply double-click on eclipse.exe

```
%C:\eclipse\eclipse.exe
```

Eclipse can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine:

```
$/usr/local/eclipse/eclipse
```

After a successful startup, if everything is fine then it will display the following result.



**\*Note:** Install m2eclipse plugin to eclipse using the following eclipse software update site

m2eclipse Plugin - <http://m2eclipse.sonatype.org/update/>

This plugin enables the developers to run maven commands within eclipse with embedded/external maven installation.

#### Step 4: Download Maven archive.

Download Maven 2.2.1 from <http://maven.apache.org/download.html>

OS	Archive name
<b>Windows</b>	apache-maven-2.0.11-bin.zip
<b>Linux</b>	apache-maven-2.0.11-bin.tar.gz
<b>Mac</b>	apache-maven-2.0.11-bin.tar.gz

### Step 5: Extract the Maven archive.

Extract the archive to the directory you wish to install Maven 2.2.1. The subdirectory apache-maven-2.2.1 will be created from the archive.

OS	Location (can be different based on your installation)
<b>Windows</b>	C:\Program Files\Apache Software Foundation\apache-maven-2.2.1
<b>Linux</b>	/usr/local/apache-maven
<b>Mac</b>	/usr/local/apache-maven

### Step 6: Set Maven environment variables.

Add M2\_HOME, M2, MAVEN\_OPTS to environment variables.

OS	Output
<b>Windows</b>	<p>Set the environment variables using system properties.</p> <p><i>M2_HOME=C:\Program Files\Apache Software Foundation\apache-maven-2.2.1</i></p> <p><i>M2=%M2_HOME%\bin</i></p> <p><i>MAVEN_OPTS=-Xms256m -Xmx512m</i></p>
<b>Linux</b>	<p>Open command terminal and set environment variables.</p> <p><i>export M2_HOME=/usr/local/apache-maven/apache-maven-2.2.1</i></p> <p><i>export M2=%M2_HOME%\bin</i></p> <p><i>export MAVEN_OPTS=-Xms256m -Xmx512m</i></p>
<b>Mac</b>	<p>Open command terminal and set environment variables.</p> <p><i>export M2_HOME=/usr/local/apache-maven/apache-maven-2.2.1</i></p> <p><i>export M2=%M2_HOME%\bin</i></p> <p><i>export MAVEN_OPTS=-Xms256m -Xmx512m</i></p>

## Step 7: Add Maven bin directory location to system path.

Now append M2 variable to System Path.

OS	Output
<b>Windows</b>	Append the string ;%M2% to the end of the system variable, Path.
<b>Linux</b>	export PATH=\$M2:\$PATH
<b>Mac</b>	export PATH=\$M2:\$PATH

## Step 8: Verify Maven installation.

Open console, execute the following **mvn** command.

OS	Task	Command
<b>Windows</b>	Open Command Console	c:\> mvn --version
<b>Linux</b>	Open Command Terminal	\$ mvn --version
<b>Mac</b>	Open Terminal	machine:~ joseph\$ mvn --version

Finally, verify the output of the above commands, which should be as shown in the following table.

OS	Output
<b>Windows</b>	Apache Maven 2.2.1 (r801777; 2009-08-07 00:46:01+0530)  Java version: 1.6.0_21  Java home: C:\Program Files\Java\jdk1.6.0_21\jre
<b>Linux</b>	Apache Maven 2.2.1 (r801777; 2009-08-07 00:46:01+0530)  Java version: 1.6.0_21  Java home: C:\Program Files\Java\jdk1.6.0_21\jre
<b>Mac</b>	Apache Maven 2.2.1 (r801777; 2009-08-07 00:46:01+0530)  Java version: 1.6.0_21  Java home: C:\Program Files\Java\jdk1.6.0_21\jre

## Step 9: Set Up Apache Tomcat.

You can download the latest version of Tomcat from <http://tomcat.apache.org/>. Once you download the installation, unpack the binary distribution into a convenient location. For example, in C:\apache-tomcat-6.0.33 on Windows, or /usr/local/apache-tomcat-6.0.33 on Linux/Unix and set CATALINA\_HOME environment variable pointing to the installation locations.

Tomcat can be started by executing the following commands on Windows machine, or you can simply double-click on startup.bat

```
%CATALINA_HOME%\bin\startup.bat
```

or

```
C:\apache-tomcat-6.0.33\bin\startup.bat
```

Tomcat can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine.

```
$CATALINA_HOME/bin/startup.sh
```

or

```
/usr/local/apache-tomcat-6.0.33/bin/startup.sh
```

After a successful startup, the default web applications included with Tomcat will be available by visiting **http://localhost:8080/**. If everything is fine, then it will display the following result.



Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site: <http://tomcat.apache.org>

Tomcat can be stopped by executing the following commands on Windows machine.

```
%CATALINA_HOME%\bin\shutdown
or
C:\apache-tomcat-5.5.29\bin\shutdown
```

Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine.

```
$CATALINA_HOME/bin/shutdown.sh
or
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```



## 3. JSF – Architecture

JSF technology is a framework for developing, building server-side User Interface Components and using them in a web application. JSF technology is based on the Model View Controller (MVC) architecture for separating logic from presentation.

### 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

The purpose of MVC design pattern is to separate model and presentation enabling developers to 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 change view layer. Controllers are used to process user actions. In this process, layer model and views may be changed.

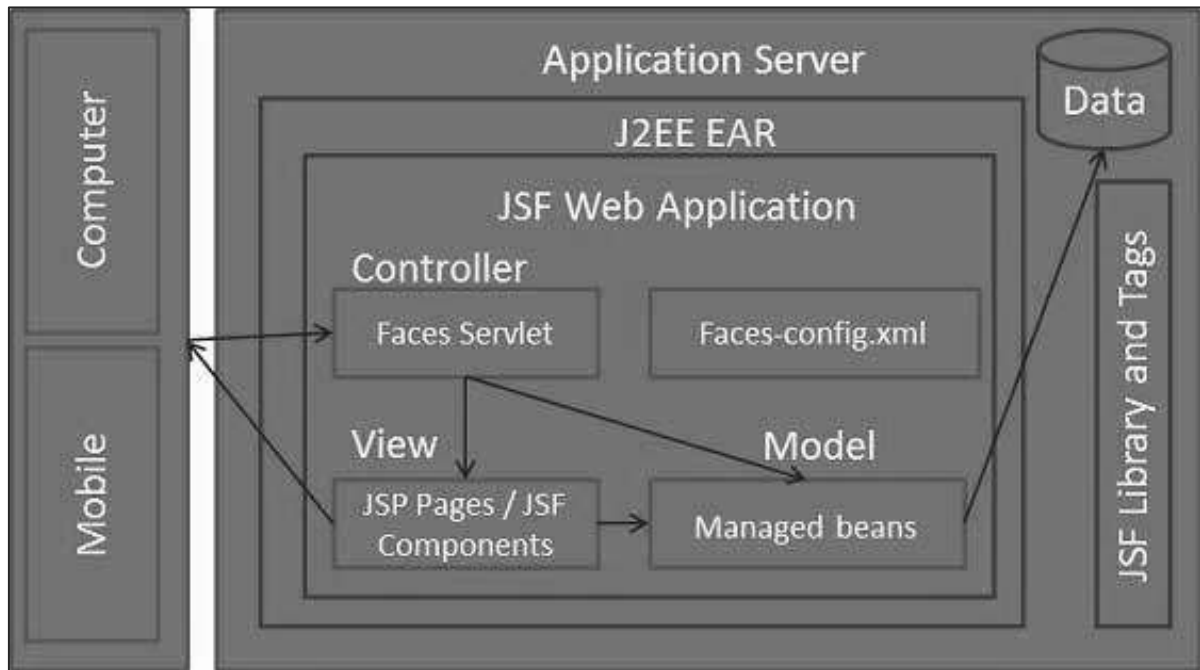
### JSF Architecture

---

JSF application is similar to any other Java technology-based web application; it runs in a Java servlet container, and contains -

- 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 handlers
- Application configuration resource file for configuring application resources



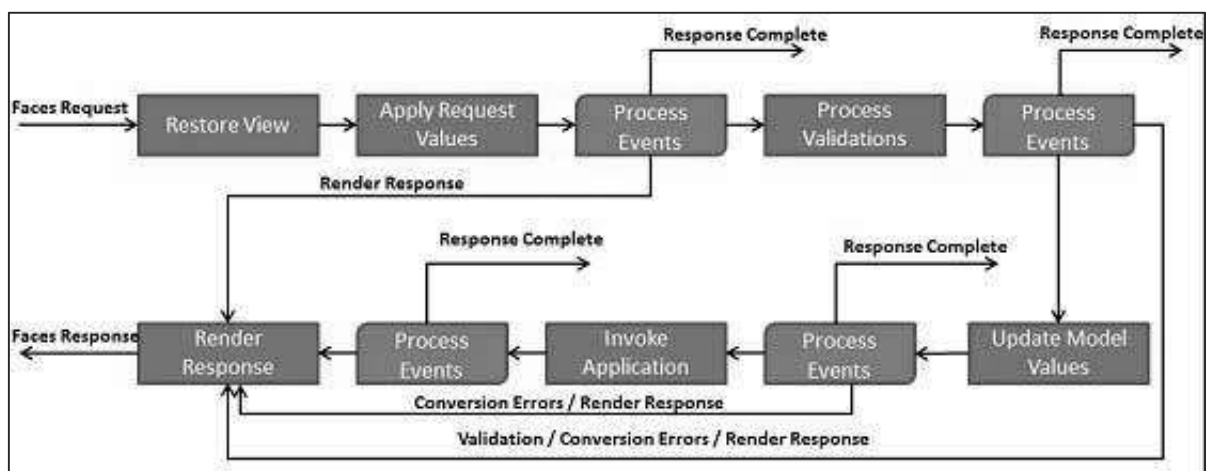
There are controllers which can be used to perform user actions. UI can be created by web page authors and the business logic can be utilized by managed beans.

JSF provides several mechanisms for rendering an individual component. It is upto the web page designer to pick the desired representation, and the application developer doesn't need to know which mechanism was used to render a JSF UI component.

## 4. JSF – Life Cycle

JSF application life cycle consists of six phases which are as follows -

- Restore view phase
- Apply request values phase; process events
- Process validations phase; process events
- Update model values phase; process events
- Invoke application phase; process events
- Render response phase



The six phases show the order in which JSF processes a form. The list shows the phases in their likely order of execution with event processing at each 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, 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 contain all the information required to process a request.

### Phase 2: Apply request values

After the component tree is created/restored, each component in the component tree uses the 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 instance, the JSF moves to the render response phase.

### **Phase 3: Process validation**

During this phase, JSF processes all validators registered on the 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, JSF adds an error message to the FacesContext instance, and the life cycle advances to the render response phase and displays 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 sets the corresponding server-side object properties to the components' local values. JSF will update the bean properties corresponding to the input component's value attribute.

If any updateModels methods called renderResponse on the current FacesContext instance, JSF moves to the render response phase.

### **Phase 5: Invoke application**

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

### **Phase 6: Render response**

During this phase, 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 JSP container executes the page. If this is not an initial request, the component tree is already built so components need not be added again. In either case, the components will render themselves as the JSP container/Application server traverses the tags in the page.

After the content of the view is rendered, the response state is saved so that subsequent requests can access it and it is available to the restore view phase.

## 5. JSF – First Application

To create a simple JSF application, we'll use maven-archetype-webapp plugin. In the following example, we'll create a maven-based web application project in C:\JSF folder.

### Create Project

Let's open command console, go the **C:\ > JSF** directory and execute the following **mvn** command.

```
C:\JSF>mvn archetype:create
-DgroupId=com.tutorialspoint.test
-DartifactId=helloworld
-DarchetypeArtifactId=maven-archetype-webapp
```

Maven will start processing and will create the complete java web application project structure.

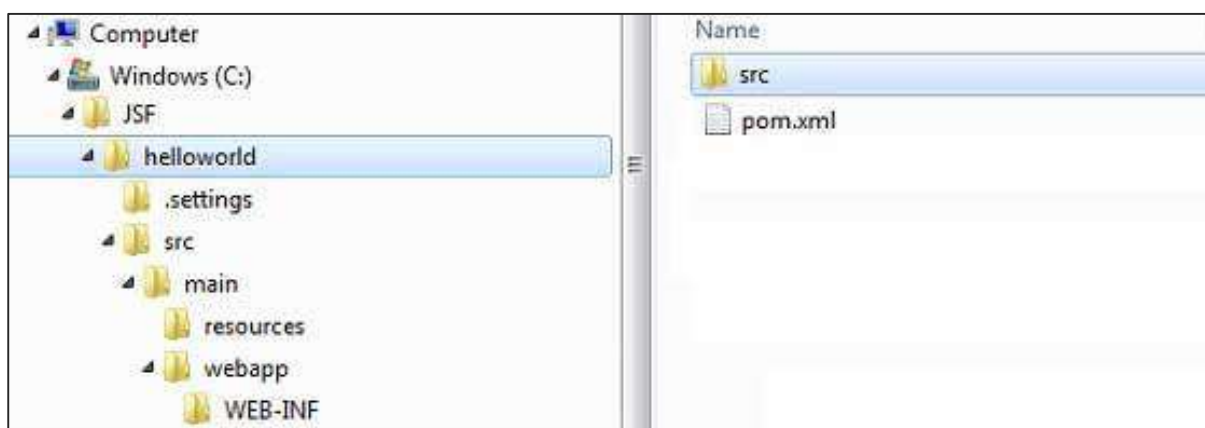
```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] -----
[INFO] Building Maven Default Project
[INFO]   task-segment: [archetype:create] (aggregator-style)
[INFO] -----
[INFO] [archetype:create {execution: default-cli}]
[INFO] Defaulting package to group ID: com.tutorialspoint.test
[INFO] artifact org.apache.maven.archetypes:maven-archetype-webapp:
checking for updates from central
[INFO] -----
[INFO] Using following parameters for creating project
from Old (1.x) Archetype: maven-archetype-webapp:RELEASE
[INFO] -----
[INFO] Parameter: groupId, Value: com.tutorialspoint.test
[INFO] Parameter: packageName, Value: com.tutorialspoint.test
[INFO] Parameter: package, Value: com.tutorialspoint.test
[INFO] Parameter: artifactId, Value: helloworld
[INFO] Parameter: basedir, Value: C:\JSF
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir:
```

```

C:\JSF\helloworld
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 7 seconds
[INFO] Finished at: Mon Nov 05 16:05:04 IST 2012
[INFO] Final Memory: 12M/84M
[INFO] -----

```

Now go to C:/JSF directory. You'll see a Java web application project created, named helloworld (as specified in artifactId). Maven uses a standard directory layout as shown in the following screenshot.



Using the above example, we can understand the following key concepts.

Folder Structure	Description
<b>helloworld</b>	Contains src folder and pom.xml
<b>src/main/webapp</b>	Contains WEB-INF folder and index.jsp page
<b>src/main/resources</b>	It contains images/properties files (In the above example, we need to create this structure manually)

## Add JSF Capability to Project

Add the following JSF dependencies.

```

<dependencies>
  <dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.1.7</version>

```

```

</dependency>

<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-impl</artifactId>
  <version>2.1.7</version>
</dependency>

</dependencies>

```

### Complete POM.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tutorialspoint.test</groupId>
  <artifactId>helloworld</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld Maven Webapp</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-api</artifactId>
      <version>2.1.7</version>
    </dependency>

```

```
<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-impl</artifactId>
  <version>2.1.7</version>
</dependency>

</dependencies>

<build>
  <finalName>helloworld</finalName>

  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.1</version>

      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>

    </plugin>

  </plugins>

</build>

</project>
```

## Prepare Eclipse Project

Let's open the command console. Go the **C:\ > JSF > helloworld** directory and execute the following **mvn** command.

```
C:\JSF\helloworld>mvn eclipse:eclipse -Dwtpversion=2.0
```

Maven will start processing, create the eclipse ready project, and will add wtp capability.



```

Downloading: http://repo.maven.apache.org/org/apache/maven/plugins/
maven-compiler-plugin/2.3.1/maven-compiler-plugin-2.3.1.pom
5K downloaded (maven-compiler-plugin-2.3.1.pom)
Downloading: http://repo.maven.apache.org/org/apache/maven/plugins/
maven-compiler-plugin/2.3.1/maven-compiler-plugin-2.3.1.jar
29K downloaded (maven-compiler-plugin-2.3.1.jar)
[INFO] Searching repository for plugin with prefix: 'eclipse'.
[INFO] -----
[INFO] Building helloworld Maven Webapp
[INFO] task-segment: [eclipse:eclipse]
[INFO] -----
[INFO] Preparing eclipse:eclipse
[INFO] No goals needed for project - skipping
[INFO] [eclipse:eclipse {execution: default-cli}]
[INFO] Adding support for WTP version 2.0.
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.
launching.JRE_CONTAINER
Downloading: http://repo.maven.apache.org/
com/sun/faces/jsf-api/2.1.7/jsf-api-2.1.7.pom
12K downloaded (jsf-api-2.1.7.pom)
Downloading: http://repo.maven.apache.org/
com/sun/faces/jsf-impl/2.1.7/jsf-impl-2.1.7.pom
10K downloaded (jsf-impl-2.1.7.pom)
Downloading: http://repo.maven.apache.org/
com/sun/faces/jsf-api/2.1.7/jsf-api-2.1.7.jar
619K downloaded (jsf-api-2.1.7.jar)
Downloading: http://repo.maven.apache.org/
com/sun/faces/jsf-impl/2.1.7/jsf-impl-2.1.7.jar
1916K downloaded (jsf-impl-2.1.7.jar)
[INFO] Wrote settings to C:\JSF\helloworld\.settings\
org.eclipse.jdt.core.prefs
[INFO] Wrote Eclipse project for "helloworld" to C:\JSF\helloworld.
[INFO]
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 6 minutes 7 seconds
[INFO] Finished at: Mon Nov 05 16:16:25 IST 2012

```

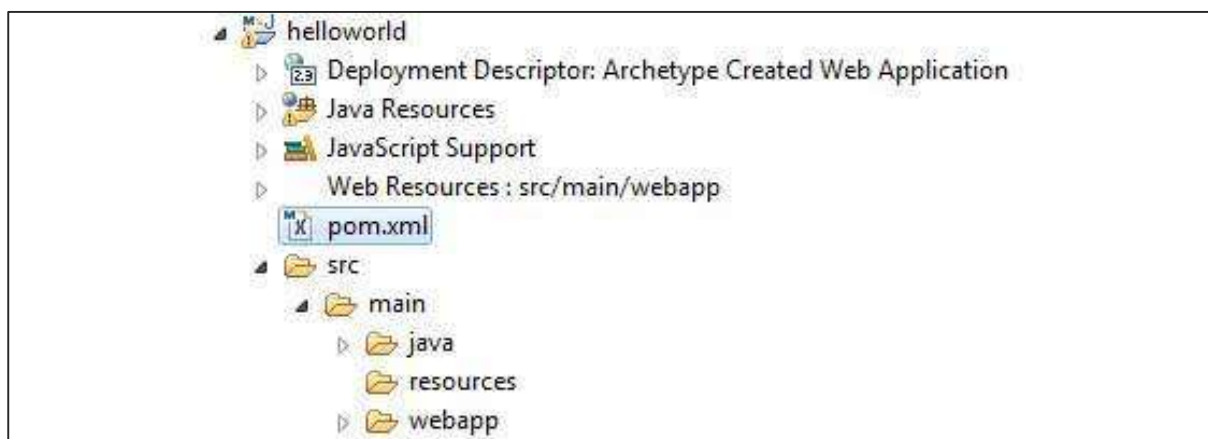
```
[INFO] Final Memory: 10M/89M
```

```
[INFO] -----
```

## Import Project in Eclipse

Following are the steps:

- Import project in eclipse using Import wizard.
- Go to **File -> Import... -> Existing project into workspace**.
- Select root directory to helloworld.
- Keep **Copy projects into workspace** to be checked.
- Click Finish button.
- Eclipse will import and copy the project in its workspace **C:\ -> Projects -> Data -> Workspace**.



## Configure Faces Servlet in web.xml

Locate web.xml in **webapp -> WEB-INF** folder and update it as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">

  <welcome-file-list>
    <welcome-file>faces/home.xhtml</welcome-file>
```

```
</welcome-file-list>

<!--
    FacesServlet is main servlet responsible to handle all request.
    It acts as central controller.
    This servlet initializes the JSF components before the JSP is displayed.
-->

<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>

<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>

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>

</web-app>
```

## Create a Managed Bean

Create a package structure under **src -> main -> java as com -> tutorialspoint -> test**. Create **HelloWorld.java** class in this package. Update the code of **HelloWorld.java** as shown below.

```
package com.tutorialspoint.test;

import javax.faces.bean.ManagedBean;

@ManagedBean(name = "helloWorld", eager = true)
public class HelloWorld {
    public HelloWorld() {
        System.out.println("HelloWorld started!");
    }

    public String getMessage() {
        return "Hello World!";
    }
}
```

## Create a JSF page

Create a page **home.xhtml** under **webapp** folder. Update the code of **home.xhtml** as shown below.

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>

<body>
    #{helloWorld.getMessage()}
</body>
</html>
```

## Build the Project

---

Following are the steps.

- Select helloworld project in eclipse
- Use Run As wizard
- Select **Run As -> Maven package**
- Maven will start building the project and will create helloworld.war under **C:\ -> Projects -> Data -> Workspace -> helloworld -> target** folder.

```
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building helloworld Maven Webapp
[INFO]
[INFO] Id: com.tutorialspoint.test:helloworld:war:1.0-SNAPSHOT
[INFO] task-segment: [package]
[INFO] -----
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] No sources to compile
[INFO] [surefire:test]
[INFO] Surefire report directory:
C:\Projects\Data\Workspace\helloworld\target\surefire-reports

-----
T E S T S
-----

There are no tests to run.

Results :
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO] [war:war]
[INFO] Packaging webapp
[INFO] Assembling webapp[helloworld] in
```

```
[C:\Projects\Data\Workspace\helloworld\target\helloworld]
[INFO] Processing war project
[INFO] Webapp assembled in[150 msecs]
[INFO] Building war:
C:\Projects\Data\Workspace\helloworld\target\helloworld.war
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 3 seconds
[INFO] Finished at: Mon Nov 05 16:34:46 IST 2012
[INFO] Final Memory: 2M/15M
[INFO] -----
```

## Deploy WAR file

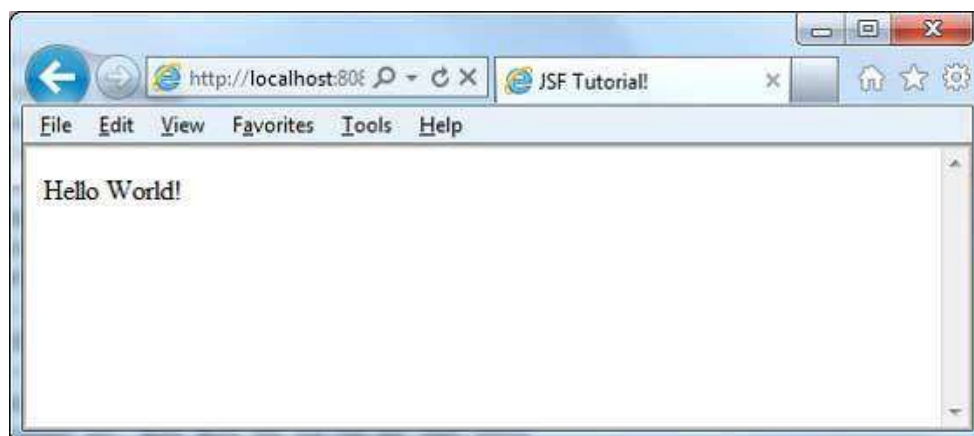
Following are the steps.

- Stop the tomcat server.
- Copy the helloworld.war file to **tomcat installation directory -> webapps folder**.
- Start the tomcat server.
- Look inside webapps directory, there should be a folder **helloworld** got created.
- Now helloworld.war is successfully deployed in Tomcat Webserver root.

## Run Application

Enter a url in web browser: **http://localhost:8080/helloworld/home.jsf** to launch the application.

Server name (localhost) and port (8080) may vary as per your tomcat configuration.



## 6. JSF – Managed Beans

Managed Bean is a regular Java Bean class registered with JSF. In other words, Managed Beans is a Java bean managed by JSF framework. Managed bean contains the getter and setter methods, business logic, or even a backing bean (a bean contains all the HTML form value).

Managed beans works as Model for UI component. Managed Bean can be accessed from JSF page.

In **JSF 1.2**, a managed bean had to register it in JSF configuration file such as faces-config.xml. From **JSF 2.0** onwards, managed beans can be easily registered using annotations. This approach keeps beans and its registration at one place hence it becomes easier to manage.

### Using XML Configuration

```
<managed-bean>
  <managed-bean-name>helloWorld</managed-bean-name>
  <managed-bean-class>com.tutorialspoint.test.HelloWorld</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
  <managed-bean-name>message</managed-bean-name>
  <managed-bean-class>com.tutorialspoint.test.Message</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

### Using Annotation

```
@ManagedBean(name = "helloWorld", eager = true)
@RequestScoped
public class HelloWorld {

    @ManagedProperty(value="#{message}")
    private Message message;

    ...
}
```

## @ManagedBean Annotation

**@ManagedBean** marks a bean to be a managed bean with the name specified in **name** attribute. If the name attribute is not specified, then the managed bean name will default to class name portion of the fully qualified class name. In our case, it would be helloWorld.

Another important attribute is **eager**. If **eager="true"** then managed bean is created before it is requested for the first time otherwise "lazy" initialization is used in which bean will be created only when it is requested.

## Scope Annotations

Scope annotations set the scope into which the managed bean will be placed. If the scope is not specified, then bean will default to request scope. Each scope is briefly discussed in the following table.

Scope	Description
<b>@RequestScoped</b>	Bean lives as long as the HTTP request-response lives. It gets created upon a HTTP request and gets destroyed when the HTTP response associated with the HTTP request is finished.
<b>@NoneScoped</b>	Bean lives as long as a single EL evaluation. It gets created upon an EL evaluation and gets destroyed immediately after the EL evaluation.
<b>@ViewScoped</b>	Bean lives as long as the user is interacting with the same JSF view in the browser window/tab. It gets created upon a HTTP request and gets destroyed once the user postbacks to a different view.
<b>@SessionScoped</b>	Bean lives as long as the HTTP session lives. It gets created upon the first HTTP request involving this bean in the session and gets destroyed when the HTTP session is invalidated.
<b>@ApplicationScoped</b>	Bean lives as long as the web application lives. It gets created upon the first HTTP request involving this bean in the application (or when the web application starts up and the <b>eager=true</b> attribute is set in <b>@ManagedBean</b> ) and gets destroyed when the web application shuts down.
<b>@CustomScoped</b>	Bean lives as long as the bean's entry in the custom Map, which is created for this scope lives.

## @ManagedProperty Annotation

JSF is a simple static Dependency Injection (DI) framework. Using **@ManagedProperty** annotation, a managed bean's property can be injected in another managed bean.



## Example Application

Let us create a test JSF application to test the above annotations for managed beans.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - Create Application</i> chapter.
2	Modify <i>HelloWorld.java</i> as explained below. Keep the rest of the files unchanged.
3	Create <i>Message.java</i> under a package <i>com.tutorialspoint.test</i> as explained below.
4	Compile and run the application to make sure business logic is working as per the requirements.
5	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
6	Launch your web application using appropriate URL as explained below in the last step.

### HelloWorld.java

```
package com.tutorialspoint.test;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.RequestScoped;

@ManagedBean(name = "helloworld", eager = true)
@RequestScoped
public class HelloWorld {

    @ManagedProperty(value="#{message}")
    private Message messageBean;
    private String message;
    public HelloWorld() {
        System.out.println("HelloWorld started!");
    }
    public String getMessage() {
        if(messageBean != null){
            message = messageBean.getMessage();
        }
    }
}
```

```

        return message;
    }
    public void setMessageBean(Message message) {
        this.messageBean = message;
    }
}

```

## Message.java

```

package com.tutorialspoint.test;

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

@ManagedBean(name = "message", eager = true)
@RequestScoped
public class Message {

    private String message = "Hello World!";
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}

```

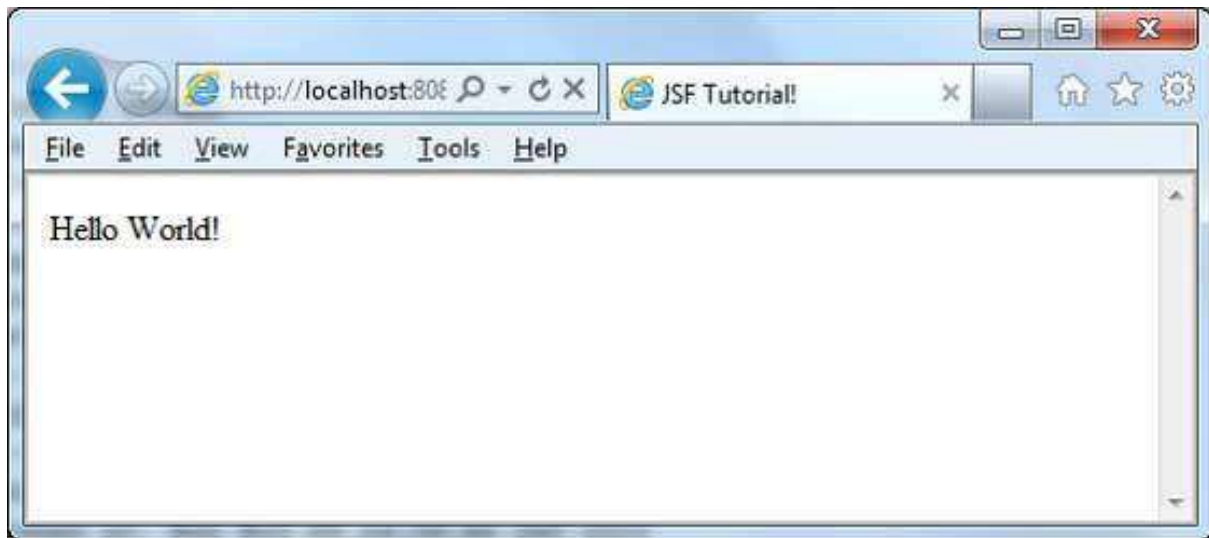
## home.xhtml

```

<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    #{helloWorld.message}
</body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.



## 7. JSF – Page Navigation

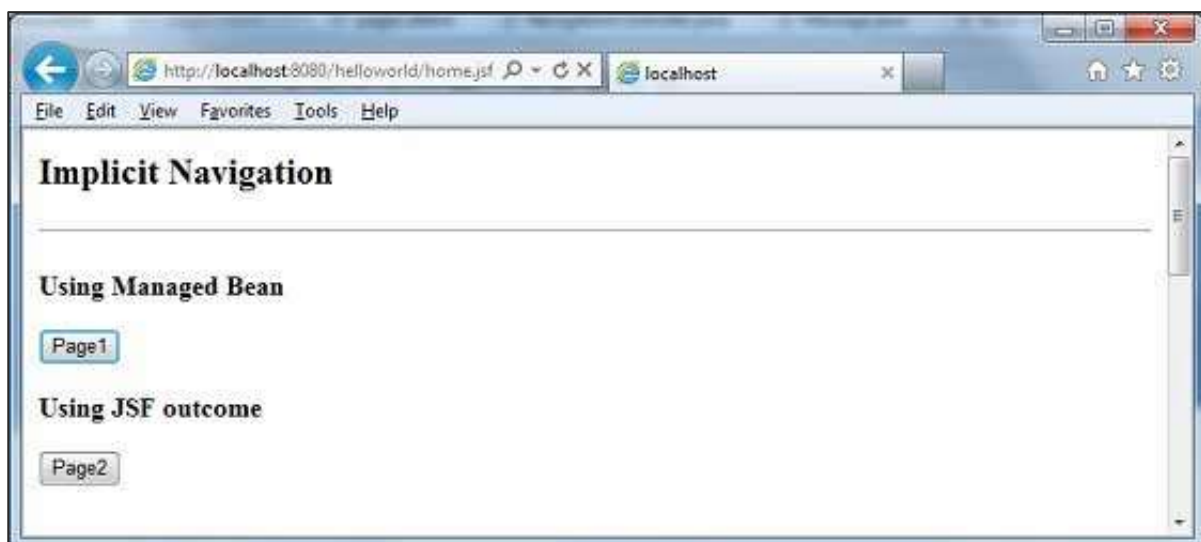
Navigation rules are those rules provided by JSF Framework that describes which view is to be shown when a button or a link is clicked.

Navigation rules can be defined in JSF configuration file named faces-config.xml. They can be defined in managed beans.

Navigation rules can contain conditions based on which the resulted view can be shown. JSF 2.0 provides implicit navigation as well in which there is no need to define navigation rules as such.

### Implicit Navigation

JSF 2.0 provides **auto view page resolver** mechanism named **implicit navigation**. In this case, you only need to put view name in **action** attribute and JSF will search the correct **view** page automatically in the deployed application.

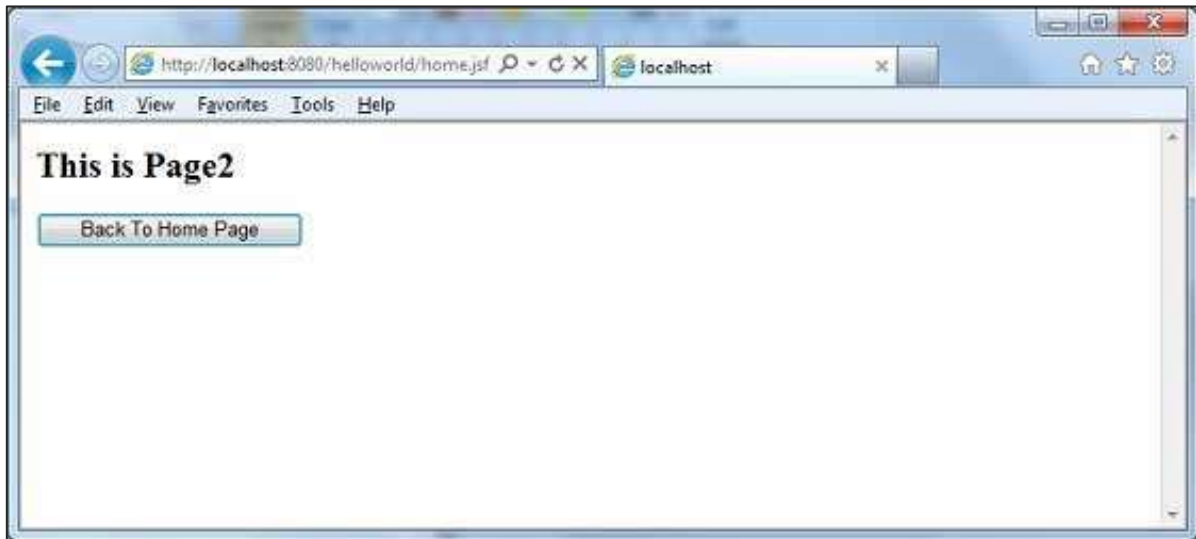


### Auto Navigation in JSF Page

Set view name in action attribute of any JSF UI Component.

```
<h:form>
  <h3>Using JSF outcome</h3>
  <h:commandButton action="page2" value="Page2" />
</h:form>
```

Here, when **Page2** button is clicked, JSF will resolve the view name, **page2** as page2.xhtml extension, and find the corresponding view file **page2.xhtml** in the current directory.



## Auto Navigation in Managed Bean

Define a method in managed bean to return a view name.

```
@ManagedBean(name = "navigationController", eager = true)
@RequestScoped
public class NavigationController implements Serializable {
    public String moveToPage1(){
        return "page1";
    }
}
```

Get view name in action attribute of any JSF UI Component using managed bean.

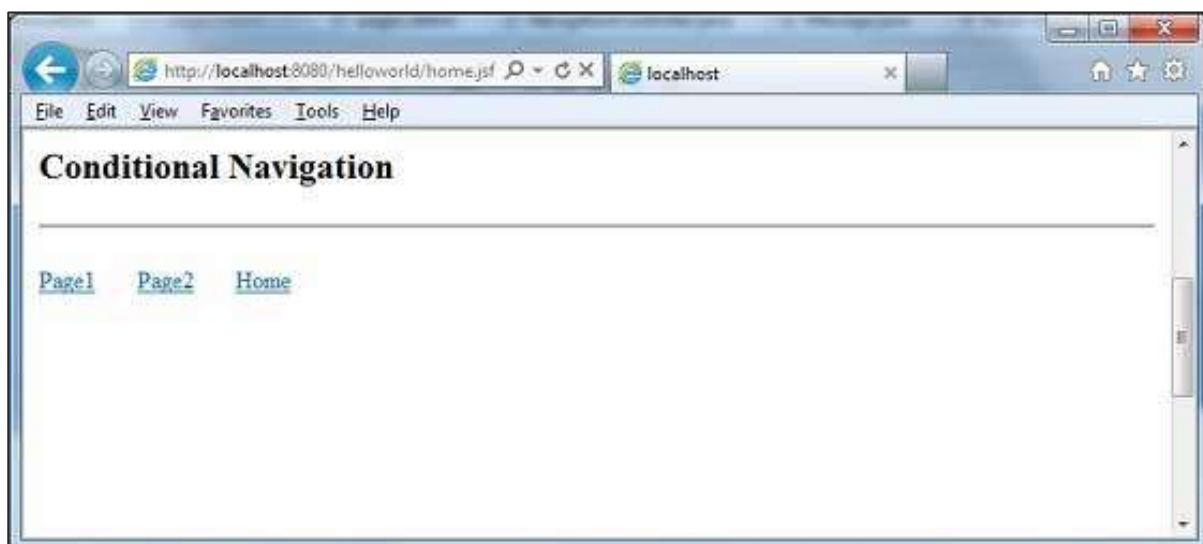
```
<h:form>
    <h3>Using Managed Bean</h3>
    <h:commandButton action="#{navigationController.moveToPage1}"
        value="Page1" />
</h:form>
```

Here, when **Page1** button is clicked, JSF will resolve the view name, **page1** as page1.xhtml extension, and find the corresponding view file **page1.xhtml** in the current directory.



## Conditional Navigation

Using managed bean, we can very easily control the navigation. Look at the following code in a managed bean.



```
@ManagedBean(name = "navigationController", eager = true)
@RequestScoped
public class NavigationController implements Serializable {

    //this managed property will read value from request parameter pageId
    @ManagedProperty(value="#{param.pageId}")
    private String pageId;

    //conditional navigation based on pageId
    //if pageId is 1 show page1.xhtml,
```

```

//if pageId is 2 show page2.xhtml
//else show home.xhtml
public String showPage(){
    if(pageId == null){
        return "home";
    }
    if(pageId.equals("1")){
        return "page1";
    }else if(pageId.equals("2")){
        return "page2";
    }else{
        return "home";
    }
}
}

```

Pass pageId as a request parameter in JSF UI Component.

```

<h:form>
    <h:commandLink action="#{navigationController.showPage}" value="Page1">
        <f:param name="pageId" value="1" />
    </h:commandLink>
    <h:commandLink action="#{navigationController.showPage}" value="Page2">
        <f:param name="pageId" value="2" />
    </h:commandLink>
    <h:commandLink action="#{navigationController.showPage}" value="Home">
        <f:param name="pageId" value="3" />
    </h:commandLink>
</h:form>

```

Here, when "Page1" button is clicked.

- JSF will create a request with parameter pageId=1
- Then JSF will pass this parameter to managed property pageId of navigationController
- Now navigationController.showPage() is called which will return view as page1 after checking the pageId
- JSF will resolve the view name, page1 as page1.xhtml extension

- Find the corresponding view file page1.xhtml in the current directory



## Resolving Navigation Based on from-action

JSF provides navigation resolution option even if managed bean different methods returns the same view name.



Look at the following code in a managed bean.

```
public String processPage1(){
    return "page";
}
public String processPage2(){
    return "page";
}
```

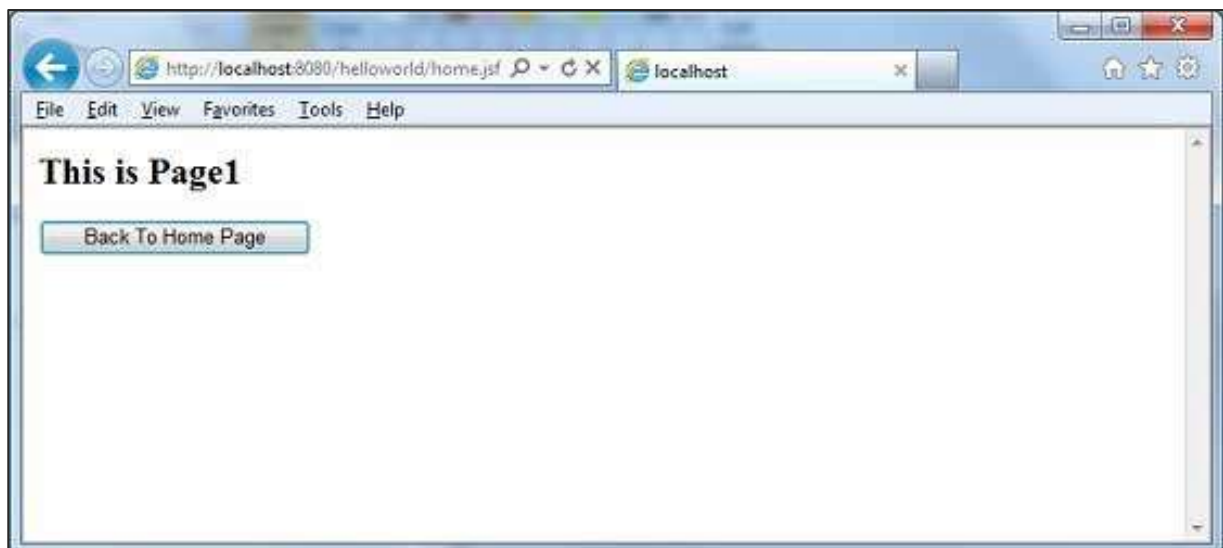


To resolve views, define the following navigation rules in **faces-config.xml**

```
<navigation-rule>
  <from-view-id>home.xhtml</from-view-id>
  <navigation-case>
    <from-action>#{navigationController.processPage1}</from-action>
    <from-outcome>page</from-outcome>
    <to-view-id>page1.jsf</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>#{navigationController.processPage2}</from-action>
    <from-outcome>page</from-outcome>
    <to-view-id>page2.jsf</to-view-id>
  </navigation-case>
</navigation-rule>
```

Here, when **Page1** button is clicked -

- **navigationController.processPage1()** is called which will return view as page
- JSF will resolve the view name, **page1** as view name is **page** and **from-action** in **faces-config** is **navigationController.processPage1**
- Find the corresponding view file **page1.xhtml** in the current directory



## Forward vs Redirect

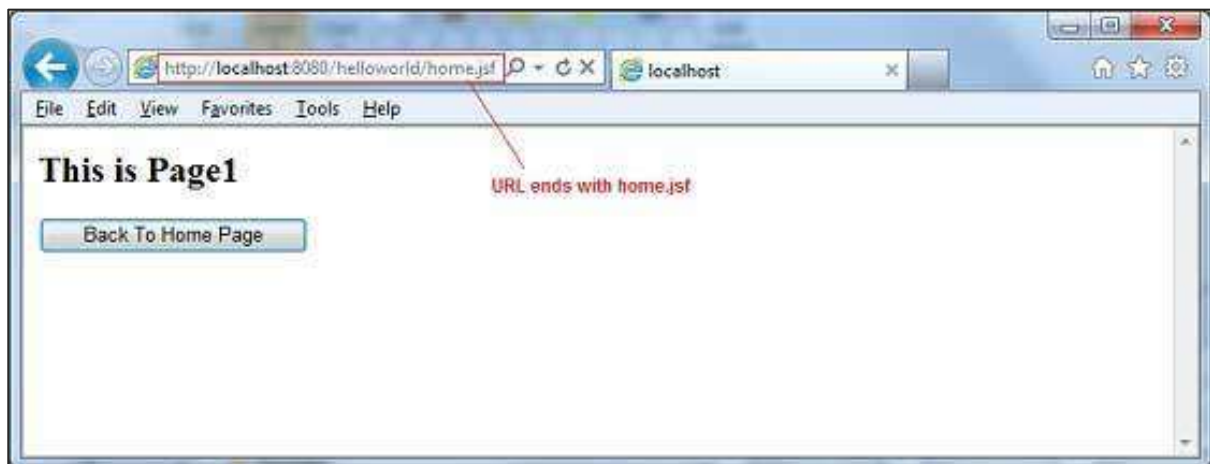
JSF by default performs a server page forward while navigating to another page and the URL of the application does not change.

To enable the page redirection, append **faces-redirect=true** at the end of the view name.



```
<h:form>
  <h3>Forward</h3>
  <h:commandButton action="page1" value="Page1" />
  <h3>Redirect</h3>
  <h:commandButton action="page1?faces-redirect=true" value="Page1" />
</h:form>
```

Here, when **Page1** button under **Forward** is clicked, you will get the following result.



Here when **Page1** button under **Redirect** is clicked, you will get the following result.



## Example Application

Let us create a test JSF application to test all of the above navigation examples.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - Create Application</i> chapter.
2	Create <i>NavigationController.java</i> under a package <i>com.tutorialspoint.test</i> as explained below.
3	Create <i>faces-config.xml</i> under a <i>WEB-INF</i> folder and updated its contents as explained below.
4	Update <i>web.xml</i> under a <i>WEB-INF</i> folder as explained below.
5	Create <i>page1.xhtml</i> and <i>page2.xhtml</i> and modify <i>home.xhtml</i> under a <i>webapp</i> folder as explained below.
6	Compile and run the application to make sure business logic is working as per the requirements.
7	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
8	Launch your web application using appropriate URL as explained below in the last step.

**NavigationController.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "navigationController", eager = true)
@RequestScoped
public class NavigationController implements Serializable {

    private static final long serialVersionUID = 1L;

    @ManagedProperty(value="#{param.pageId}")
    private String pageId;

    public String moveToPage1(){
        return "page1";
    }

    public String moveToPage2(){
        return "page2";
    }

    public String moveToHomePage(){
        return "home";
    }

    public String processPage1(){
        return "page";
    }

    public String processPage2(){
        return "page";
    }
}
```

```

public String showPage(){
    if(pageId == null){
        return "home";
    }
    if(pageId.equals("1")){
        return "page1";
    }else if(pageId.equals("2")){
        return "page2";
    }else{
        return "home";
    }
}

public String getPageId() {
    return pageId;
}

public void setPageId(String pageId) {
    this.pageId = pageId;
}
}

```

### faces-config.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<faces-config
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
    version="2.0">
    <navigation-rule>
        <from-view-id>home.xhtml</from-view-id>
        <navigation-case>
            <from-action>#{navigationController.processPage1}</from-action>
            <from-outcome>page</from-outcome>
            <to-view-id>page1.jsf</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-action>#{navigationController.processPage2}</from-action>

```

```

        <from-outcome>page</from-outcome>
        <to-view-id>page2.jsf</to-view-id>
    </navigation-case>
</navigation-rule>
</faces-config>

```

## web.xml

```

<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
<display-name>Archetype Created Web Application</display-name>

<context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
</context-param>
<context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
</context-param>
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
</web-app>

```

**page1.xhtml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
    <h:body>
        <h2>This is Page1</h2>
        <h:form>
            <h:commandButton action="home?faces-redirect=true"
                value="Back To Home Page" />
        </h:form>
    </h:body>
</html>
```

**page2.xhtml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
    <h:body>
        <h2>This is Page2</h2>
        <h:form>
            <h:commandButton action="home?faces-redirect=true"
                value="Back To Home Page" />
        </h:form>
    </h:body>
</html>
```

**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">

  <h:body>
    <h2>Implicit Navigation</h2>
    <hr />
    <h:form>
      <h3>Using Managed Bean</h3>
      <h:commandButton action="#{navigationController.moveToPage1}"
        value="Page1" />
      <h3>Using JSF outcome</h3>
      <h:commandButton action="page2" value="Page2" />
    </h:form>
    <br/>
    <h2>Conditional Navigation</h2>
    <hr />
    <h:form>
      <h:commandLink action="#{navigationController.showPage}"
        value="Page1">
        <f:param name="pageId" value="1" />
      </h:commandLink>

      <h:commandLink action="#{navigationController.showPage}"
        value="Page2">
        <f:param name="pageId" value="2" />
      </h:commandLink>

      <h:commandLink action="#{navigationController.showPage}"
        value="Home">
        <f:param name="pageId" value="3" />
      </h:commandLink>
    </h:form>
    <br/>

```

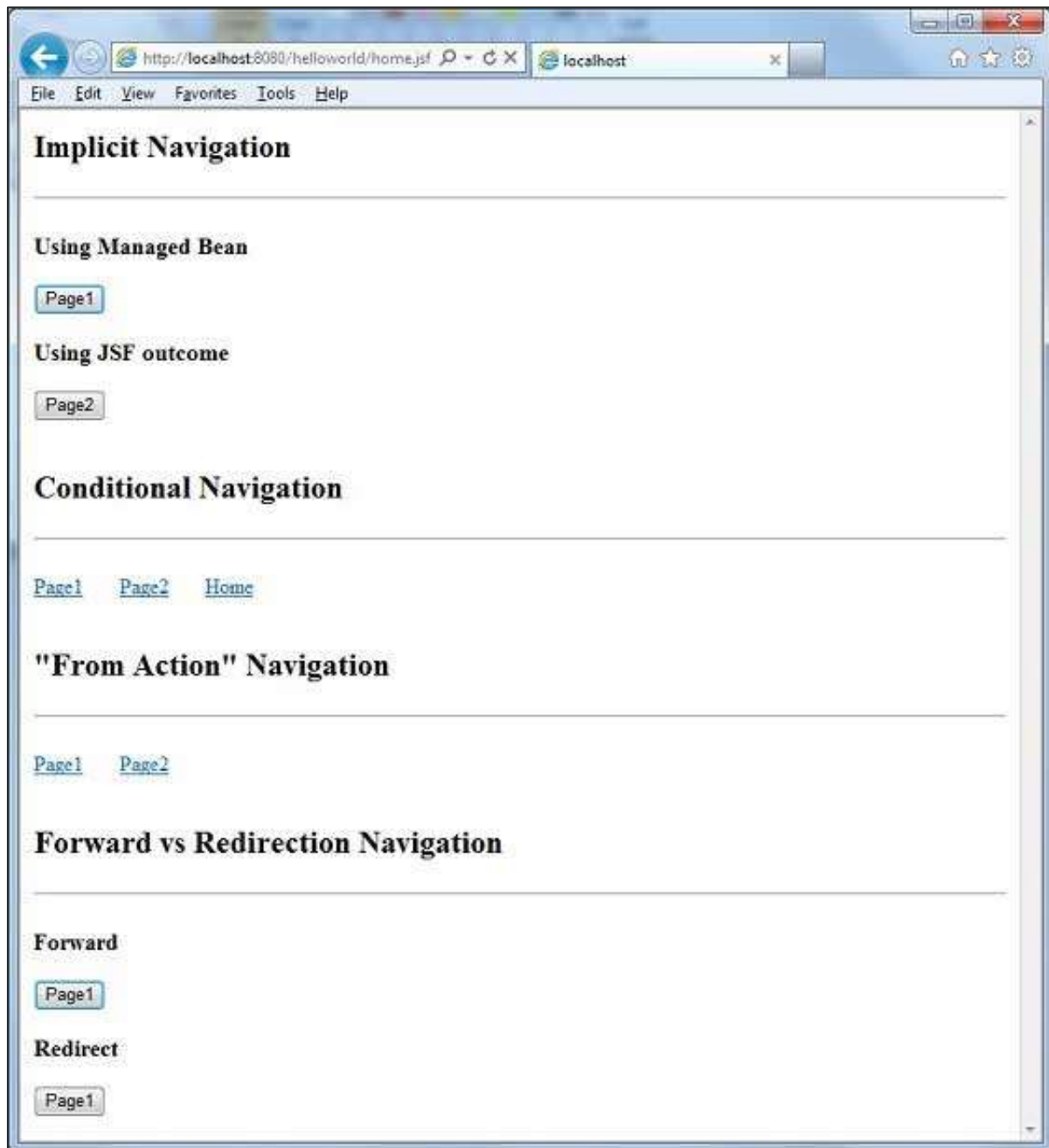


```
<h2>"From Action" Navigation</h2>
<hr />
<h:form>
    <h:commandLink action="#{navigationController.processPage1}"
        value="Page1" />

    <h:commandLink action="#{navigationController.processPage2}"
        value="Page2" />

</h:form>
<br/>
<h2>Forward vs Redirection Navigation</h2>
<hr />
<h:form>
    <h3>Forward</h3>
    <h:commandButton action="page1" value="Page1" />
    <h3>Redirect</h3>
    <h:commandButton action="page1?faces-redirect=true"
        value="Page1" />
</h:form>
</h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.



## 8. JSF – Basic Tags

In this chapter, you will learn about various types of basic JSF 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 the important *Basic Tags* in JSF 2.0.

Sr. No.	Tag & Description
1	<b><u>h:inputText</u></b> Renders a HTML input of type="text", text box.
2	<b><u>h:inputSecret</u></b> Renders a HTML input of type="password", text box.
3	<b><u>h:inputTextarea</u></b> Renders a HTML textarea field.
4	<b><u>h:inputHidden</u></b> Renders a HTML input of type="hidden".
5	<b><u>h:selectBooleanCheckbox</u></b> Renders a single HTML check box.
6	<b><u>h:selectManyCheckbox</u></b> Renders a group of HTML check boxes.
7	<b><u>h:selectOneRadio</u></b> Renders a single HTML radio button.
8	<b><u>h:selectOneListbox</u></b> Renders a HTML single list box.

9	<b><u>h:selectManyListbox</u></b> Renders a HTML multiple list box.
10	<b><u>h:selectOneMenu</u></b> Renders a HTML combo box.
11	<b><u>h:outputText</u></b> Renders a HTML text.
12	<b><u>h:outputFormat</u></b> Renders a HTML text. It accepts parameters.
13	<b><u>h:graphicImage</u></b> Renders an image.
14	<b><u>h:outputStylesheet</u></b> Includes a CSS style sheet in HTML output.
15	<b><u>h:outputScript</u></b> Includes a script in HTML output.
16	<b><u>h:commandButton</u></b> Renders a HTML input of type="submit" button.
17	<b><u>h:Link</u></b> Renders a HTML anchor.
18	<b><u>h:commandLink</u></b> Renders a HTML anchor.
19	<b><u>h:outputLink</u></b> Renders a HTML anchor.
20	<b><u>h:panelGrid</u></b> Renders an HTML Table in form of grid.
21	<b><u>h:message</u></b> Renders message for a JSF UI Component.

22	<b><u>h:messages</u></b> Renders all message for JSF UI Components.
23	<b><u>f:param</u></b> Pass parameters to JSF UI Component.
24	<b><u>f:attribute</u></b> Pass attribute to a JSF UI Component.
25	<b><u>f:setPropertyActionListener</u></b> Sets value of a managed bean's property.

## h:inputText

The h:inputText tag renders an HTML input element of the type "text".

### JSF Tag

```
<h:inputText value="Hello World!" />
```

### Rendered Output

```
<input type="text" name="j_idt6:j_idt8" value="Hello World!" />
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name

5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b> .
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>border</b> Pixel value for an element's border width
15	<b>charset</b> Character encoding for a linked resource
16	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon

17	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left).
18	<b>disabled</b> Disabled state of an input element or button
19	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
20	<b>lang</b> Base language of an element's attributes and text
21	<b>maxlength</b> Maximum number of characters for text fields
22	<b>readonly</b> Read-only state of an input field; the text can be selected in a readonly field but not edited
23	<b>style</b> Inline style information
24	<b>tabindex</b> Numerical value specifying a tab index
25	<b>target</b> The name of a frame in which a document is opened
26	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
27	<b>type</b> Type of a link; for example, <b>stylesheet</b>

28	<b>width</b> Width of an element
29	<b>onblur</b> Element loses focus
30	<b>onchange</b> Element's value changes
31	<b>onclick</b> Mouse button is clicked over the element
32	<b>ondblclick</b> Mouse button is double-clicked over the element
33	<b>onfocus</b> Element receives focus
34	<b>onkeydown</b> Key is pressed
35	<b>onkeypress</b> Key is pressed and subsequently released
36	<b>onkeyup</b> Key is released
37	<b>onmousedown</b> Mouse button is pressed over the element
38	<b>onmousemove</b> Mouse moves over the element
39	<b>onmouseout</b> Mouse leaves the element's area



40	<b>onmouseover</b> Mouse moves onto an element
41	<b>onmouseup</b> Mouse button is released
42	<b>onreset</b> Form is reset
43	<b>onselect</b> Text is selected in an input field
44	<b>immediate</b> Process validation early in the life cycle

## Example Application

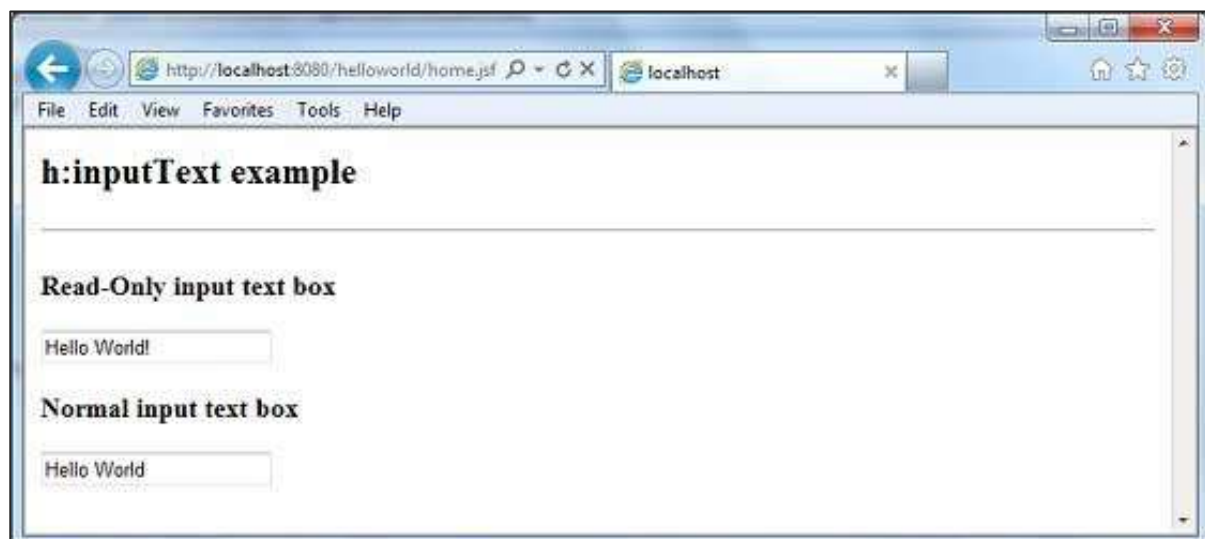
Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

**home.xhtml**

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:inputText example</h2>
    <hr />
    <h:form>
        <h3>Read-Only input text box</h3>
        <h:inputText value="Hello World!" readonly="true"/>
        <h3>Read-Only input text box</h3>
        <h:inputText value="Hello World"/>
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:inputSecret

The h:inputSecret tag renders an HTML input element of the type "password".

### JSF Tag

```
<h:inputSecret value="password" />
```

### Rendered Output

```
<input type="password" name="j_idt12:j_idt16" value="password" />
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field

10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b>  Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>border</b> Pixel value for an element's border width
15	<b>charset</b> Character encoding for a linked resource
16	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
17	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left).
18	<b>disabled</b> Disabled state of an input element or button
19	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
20	<b>lang</b> Base language of an element's attributes and text
21	<b>maxlength</b> Maximum number of characters for text fields

22	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited
23	<b>style</b> Inline style information
24	<b>tabindex</b> Numerical value specifying a tab index
25	<b>target</b> The name of a frame in which a document is opened
26	<b>title</b>  A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
27	<b>type</b> Type of a link; for example, <b>stylesheet</b>
28	<b>width</b> Width of an element
29	<b>onblur</b> Element loses focus
30	<b>onchange</b> Element's value changes
31	<b>onclick</b> Mouse button is clicked over the element
32	<b>ondblclick</b> Mouse button is double-clicked over the element
33	<b>onfocus</b> Element receives focus
34	<b>onkeydown</b> Key is pressed

35	<b>onkeypress</b> Key is pressed and subsequently released
36	<b>onkeyup</b> Key is released
37	<b>onmousedown</b> Mouse button is pressed over the element
38	<b>onmousemove</b> Mouse moves over the element
39	<b>onmouseout</b> Mouse leaves the element's area
40	<b>onmouseover</b> Mouse moves onto an element
41	<b>onmouseup</b> Mouse button is released
42	<b>onreset</b> Form is reset
43	<b>onselect</b> Text is selected in an input field
44	<b>immediate</b> Process validation early in the life cycle
45	<b>redisplay</b> when true, the input field's value is redisplayed when the web page is reloaded

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:inputSecret example</h2>
    <hr />
    <h:form>
        <h3>Read-Only input password box</h3>
        <h:inputSecret value="password" readonly="true"/>
    </h:form>
</body>
</html>
```

```

    <h3>Read-Only input text box</h3>
    <h:inputText value="password"/>
  </h:form>
</body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:inputTextarea

The h:inputText tag renders an HTML input element of the type "text".

## JSF Tag

```

<h:inputTextarea row="10" col="10" value="Hello World!
Everything is fine!" readonly="true"/>

```

## Rendered Output

```

<textarea name="j_idt18:j_idt20" readonly="readonly">
  Hello World! Everything is fine!</textarea>

```

## Tag Attributes



Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b>

	Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b> .
13	<b>cols</b> Number of columns
14	<b>border</b> Pixel value for an element's border width
15	<b>charset</b> Character encoding for a linked resource
16	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
17	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
18	<b>disabled</b> Disabled state of an input element or button
19	<b>hreflang</b>  Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b> .
20	<b>lang</b> Base language of an element's attributes and text
21	<b>rows</b> Number of rows
22	<b>readonly</b>  Read-only state of an input field; the text can be selected in a readonly field but not edited

23	<b>style</b> Inline style information
24	<b>tabindex</b> Numerical value specifying a tab index
25	<b>target</b> The name of a frame in which a document is opened
26	<b>title</b>  A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
27	<b>type</b> Type of a link; for example, <b>stylesheet</b>
28	<b>width</b> Width of an element
29	<b>onblur</b> Element loses focus
30	<b>onchange</b> Element's value changes
31	<b>onclick</b> Mouse button is clicked over the element
32	<b>ondblclick</b> Mouse button is double-clicked over the element
33	<b>onfocus</b> Element receives focus
34	<b>onkeydown</b> Key is pressed

35	<b>onkeypress</b> Key is pressed and subsequently released
36	<b>onkeyup</b> Key is released
37	<b>onmousedown</b> Mouse button is pressed over the element
38	<b>onmousemove</b> Mouse moves over the element
39	<b>onmouseout</b> Mouse leaves the element's area
40	<b>onmouseover</b> Mouse moves onto an element
41	<b>onmouseup</b> Mouse button is released
42	<b>onreset</b> Form is reset
43	<b>onselect</b> Text is selected in an input field
44	<b>immediate</b> Process validation early in the life cycle

## Example Application

Let us create a test JSF application to test the above tag.

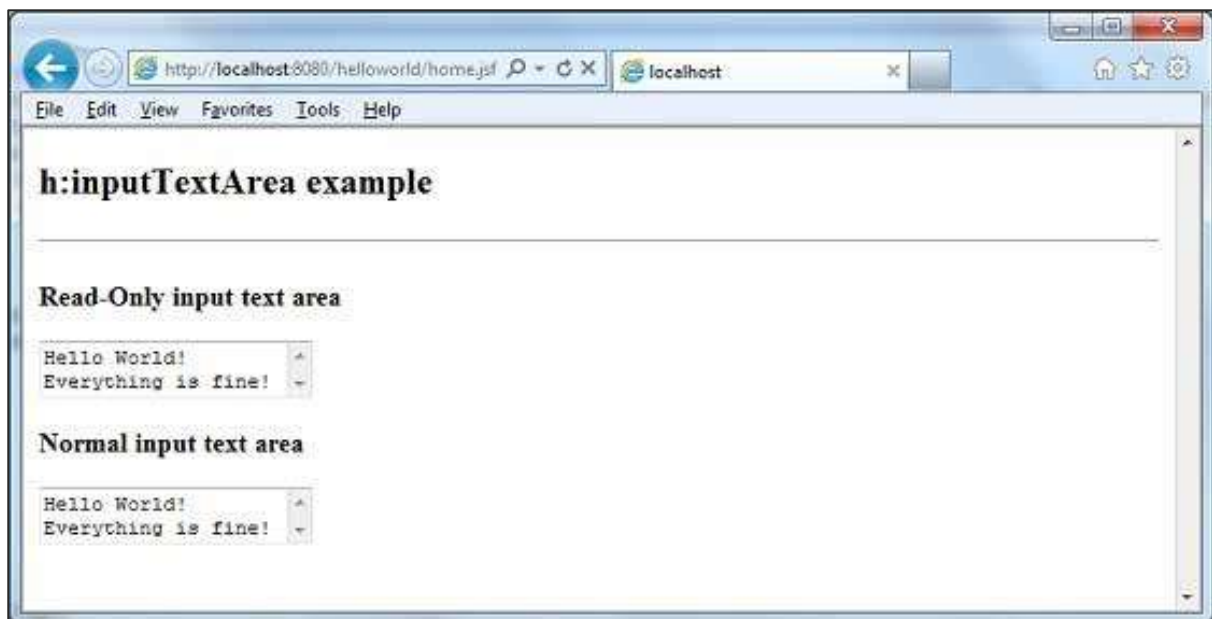
Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.

2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:inputTextArea example</h2>
    <hr />
    <h:form>
        <h3>Read-Only input text area</h3>
        <h:inputTextarea row="10" col="10" value="Hello World!
            <br/> Everything is fine!" readonly="true"/>
        <h3>Normal input text area</h3>
        <h:inputTextarea value="Hello World! <br/> Everything is fine!"/>
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:inputHidden

The `h:inputHidden` tag renders an HTML input element of the type "hidden".

### JSF Tag

```
<h:inputHidden value="Hello World" id="hiddenField" />
```

### Rendered Output

```
<input id="jsfForm:hiddenField" type="hidden" name="jsfForm:hiddenField"
value="Hello World" />
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering

4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b>  A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>cols</b> Number of columns
14	<b>border</b> Pixel value for an element's border width
15	<b>charset</b> Character encoding for a linked resource

16	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
17	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left).
18	<b>disabled</b> Disabled state of an input element or button
19	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
20	<b>lang</b> Base language of an element's attributes and text
21	<b>rows</b> Number of rows
22	<b>readonly</b> Read-only state of an input field; the text can be selected in a readonly field but not edited
23	<b>style</b> Inline style information
24	<b>tabindex</b> Numerical value specifying a tab index
25	<b>target</b> The name of a frame in which a document is opened
26	<b>title</b>



	A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
27	<b>type</b> Type of a link; for example, <b>stylesheet</b>
28	<b>width</b> Width of an element
29	<b>onblur</b> Element loses focus
30	<b>onchange</b> Element's value changes
31	<b>onclick</b> Mouse button is clicked over the element
32	<b>ondblclick</b> Mouse button is double-clicked over the element
33	<b>onfocus</b> Element receives focus
34	<b>onkeydown</b> Key is pressed
35	<b>onkeypress</b> Key is pressed and subsequently released
36	<b>onkeyup</b> Key is released
37	<b>onmousedown</b> Mouse button is pressed over the element
38	<b>onmousemove</b> Mouse moves over the element

39	<b>onmouseout</b> Mouse leaves the element's area
40	<b>onmouseover</b> Mouse moves onto an element
41	<b>onmouseup</b> Mouse button is released
42	<b>onreset</b> Form is reset
43	<b>onselect</b> Text is selected in an input field
44	<b>immediate</b> Process validation early in the life cycle

## Example Application

Let us create a test JSF application to test the above tag.

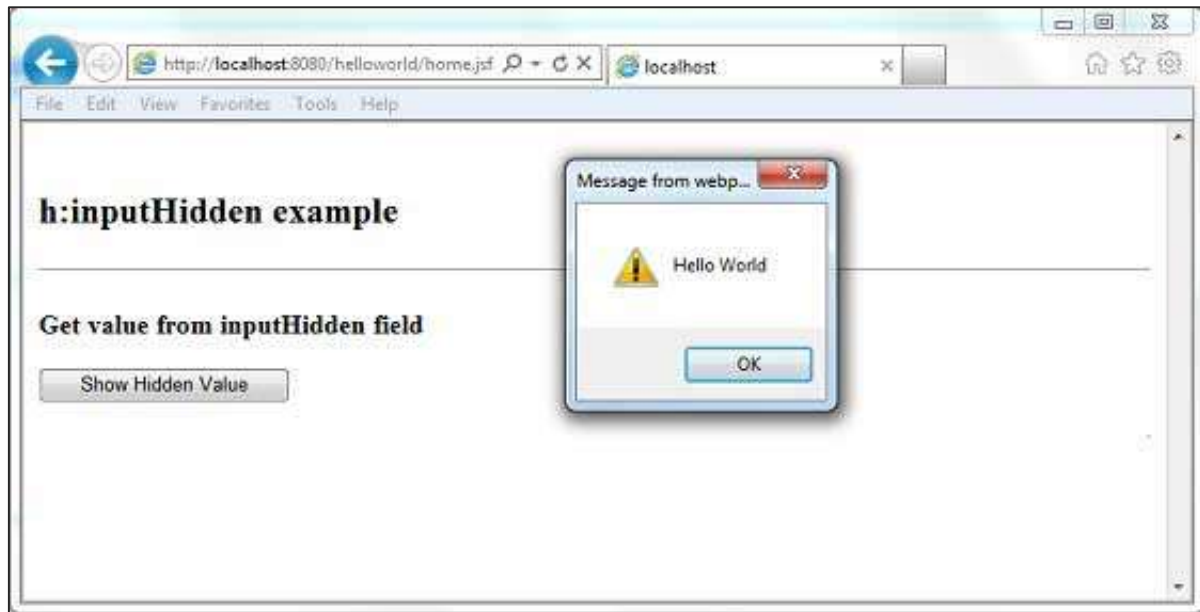
Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	

	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
    <h:head>
        <script type="text/javascript">
            function showHiddenValue(){
                alert(document.getElementById('jsfForm:hiddenField').value);
            }
        </script>
    </h:head>
</head>
<body>
<h2>h:inputHidden example</h2>
<hr />
    <h:form id="jsfForm">
        <h3>Get value from inputHidden field</h3>
        <h:inputHidden value="Hello World" id="hiddenField" />
        <h:commandButton value="Show Hidden Value" onclick="showHiddenValue()" />
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.



## h:selectBooleanCheckbox

The h:selectBooleanCheckbox tag renders an HTML input element of the type "checkbox".

### JSF Tag

```
<h:selectBooleanCheckbox value="Remember Me" id="chkRememberMe" />
```

### Rendered Output

```
<input id="jsfForm1:chkRememberMe" type="checkbox"
  name="jsfForm1:chkRememberMe" checked="checked" />
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component

2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form

12	<p><b>accept-charset</b></p> <p>Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b></p>
13	<p><b>alt</b></p> <p>Alternative text for nontextual elements such as images or applets</p>
14	<p><b>charset</b></p> <p>Character encoding for a linked resource</p>
15	<p><b>coords</b></p> <p>Coordinates for an element whose shape is a rectangle, circle, or polygon</p>
16	<p><b>dir</b></p> <p>Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left).</p>
17	<p><b>disabled</b></p> <p>Disabled state of an input element or button</p>
18	<p><b>hreflang</b></p> <p>Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>.</p>
19	<p><b>lang</b></p> <p>Base language of an element's attributes and text</p>
20	<p><b>maxlength</b></p> <p>Maximum number of characters for text fields</p>
21	<p><b>readonly</b></p> <p>Read-only state of an input field; text can be selected in a readonly field but not edited</p>

22	<b>rel</b> Relationship between the current document and a link specified with the <b>href</b> attribute
23	<b>rev</b> Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types.
24	<b>rows</b> Number of visible rows in a text area. <b>h:dataTable</b> has a <b>rows</b> attribute, but it's not an HTML pass-through attribute.
25	<b>shape</b> Shape of a region. Valid values: <b>default</b> , <b>rect</b> , <b>circle</b> , <b>poly</b> . (default signifies the entire region)
26	<b>style</b> Inline style information
27	<b>tabindex</b> Numerical value specifying a tab index
28	<b>target</b> The name of a frame in which a document is opened
29	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
30	<b>type</b> Type of a link; for example, <b>stylesheet</b>
31	<b>width</b> Width of an element

32	<b>onblur</b> Element loses focus
33	<b>onchange</b> Element's value changes
34	<b>onclick</b> Mouse button is clicked over the element
35	<b>ondblclick</b> Mouse button is double-clicked over the element
36	<b>onfocus</b> Element receives focus
37	<b>onkeydown</b> Key is pressed
38	<b>onkeypress</b> Key is pressed and subsequently released
39	<b>onkeyup</b> Key is released
40	<b>onmousedown</b> Mouse button is pressed over the element
41	<b>onmousemove</b> Mouse moves over the element
42	<b>onmouseout</b> Mouse leaves the element's area
43	<b>onmouseover</b>



	Mouse moves onto an element
44	<b>onmouseup</b> Mouse button is released
45	<b>onreset</b> Form is reset
46	<b>onselect</b> Text is selected in an input field

## Example Application

Let us create a test JSF application to test the above tag.

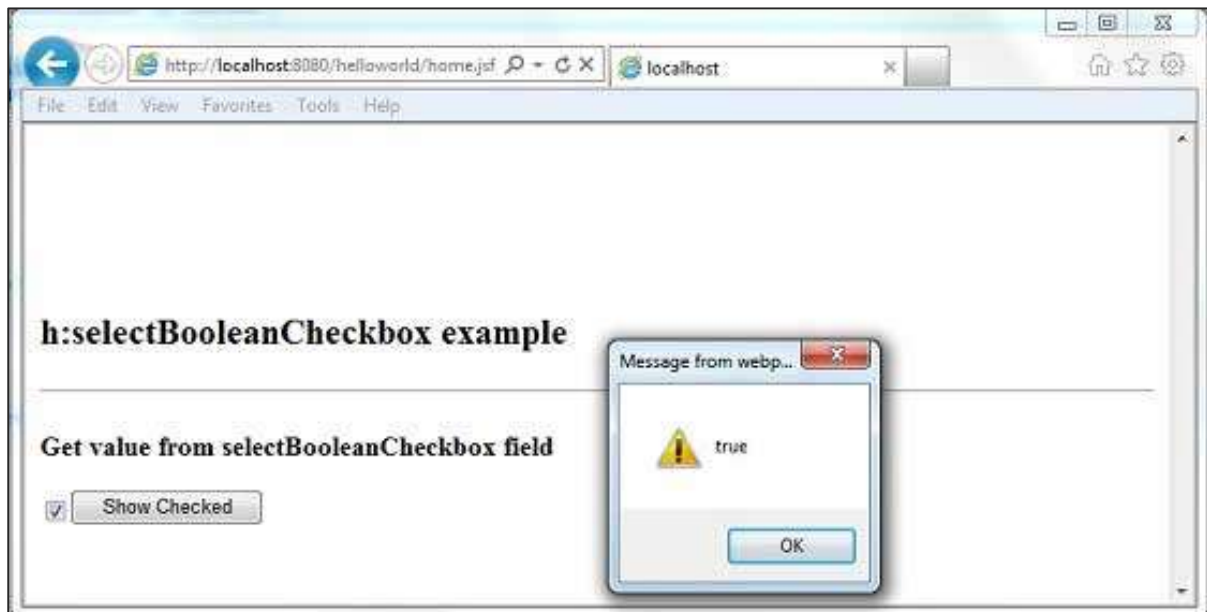
Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

5	Launch your web application using appropriate URL as explained below in the last step.
---	--

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
    <h:head>
        <script type="text/javascript">
            function showCheckedValue(){
                alert(document.getElementById('jsfForm1:chkRememberMe').checked);
            }
        </script>
    </h:head>
</head>
<h2>h:selectBooleanCheckbox example</h2>
<hr />
    <h:form id="jsfForm1">
        <h3>Get value from selectBooleanCheckbox field</h3>
        <h:selectBooleanCheckbox value="Remember Me" id="chkRememberMe" />
        <h:commandButton value="Show Checked" onclick="showCheckedValue()" />
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:selectManyCheckbox

The h:selectManyCheckbox tag renders a set of HTML input element of type "checkbox", and format it with HTML table and label tags.

### JSF Tag

```
<h:selectManyCheckbox value="#{userData.data}">
  <f:selectItem itemValue="1" itemLabel="Item 1" />
  <f:selectItem itemValue="2" itemLabel="Item 2" />
</h:selectManyCheckbox>
```

### Rendered Output

```
<table>
  <tr>
    <td><input name="j_idt6:j_idt8" id="j_idt6:j_idt8:0" value="1"
      type="checkbox" checked="checked" />
      <label for="j_idt6:j_idt8:0" class=""> Item 1</label>
    </td>
    <td><input name="j_idt6:j_idt8" id="j_idt6:j_idt8:1" value="2"
      type="checkbox" checked="checked" />
      <label for="j_idt6:j_idt8:1" class=""> Item 2</label>
    </td>
  </tr>
</table>
```

```

    </td>
  </tr>
</table>

```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component

9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>charset</b> Character encoding for a linked resource
15	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
16	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
17	<b>disabled</b> Disabled state of an input element or button
18	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>

19	<b>lang</b> Base language of an element's attributes and text
20	<b>maxlength</b> Maximum number of characters for text fields
21	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited
22	<b>rel</b> Relationship between the current document and a link specified with the <b>href</b> attribute
23	<b>rev</b> Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types
24	<b>rows</b> Number of visible rows in a text area. <b>h:dataTable</b> has a <b>rows</b> attribute, but it's not an HTML pass-through attribute
25	<b>shape</b> Shape of a region. Valid values: <b>default</b> , <b>rect</b> , <b>circle</b> , <b>poly</b> . (default signifies the entire region)
26	<b>style</b> Inline style information
27	<b>tabindex</b> Numerical value specifying a tab index

28	<b>target</b> The name of a frame in which a document is opened
29	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
30	<b>type</b> Type of a link; for example, <b>stylesheet</b>
31	<b>width</b> Width of an element
32	<b>onblur</b> Element loses focus
33	<b>onchange</b> Element's value changes
34	<b>onclick</b> Mouse button is clicked over the element
35	<b>ondblclick</b> Mouse button is double-clicked over the element
36	<b>onfocus</b> Element receives focus
37	<b>onkeydown</b> Key is pressed
38	<b>onkeypress</b> Key is pressed and subsequently released

39	<b>onkeyup</b> Key is released
40	<b>onmousedown</b> Mouse button is pressed over the element
41	<b>onmousemove</b> Mouse moves over the element
42	<b>onmouseout</b> Mouse leaves the element's area
43	<b>onmouseover</b> Mouse moves onto an element
44	<b>onmouseup</b> Mouse button is released
45	<b>onreset</b> Form is reset
46	<b>onselect</b> Text is selected in an input field
47	<b>disabledClass</b> CSS class for disabled elements
48	<b>enabledClass</b> CSS class for enabled elements
49	<b>layout</b>  Specification for how elements are laid out: <code>lineDirection</code> (horizontal) or <code>pageDirection</code> (vertical)



50	<b>border</b> Border of the element
----	--

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```

package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String[] data = {"1","2","3"};

    public String[] getData() {
        return data;
    }

    public void setData(String[] data) {
        this.data = data;
    }
}

```

### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
        <h2>h:selectManyCheckbox example</h2>

```

```

<hr />
<h:form>
<h3>Mutiple checkboxes</h3>
<h:selectManyCheckbox value="#{userData.data}">
    <f:selectItem itemValue="1" itemLabel="Item 1" />
    <f:selectItem itemValue="2" itemLabel="Item 2" />
    <f:selectItem itemValue="3" itemLabel="Item 3" />
    <f:selectItem itemValue="4" itemLabel="Item 4" />
    <f:selectItem itemValue="5" itemLabel="Item 5" />
</h:selectManyCheckbox>
<h:commandButton value="Submit" action="result" />
</h:form>
</h:body>
</html>

```

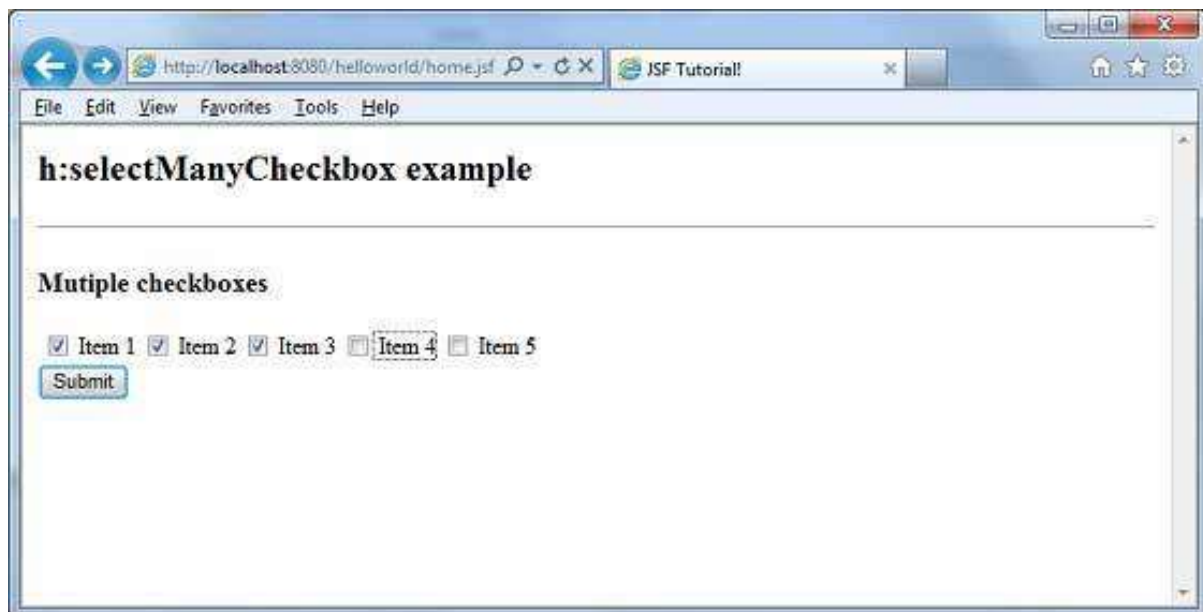
### result.xhtml

```

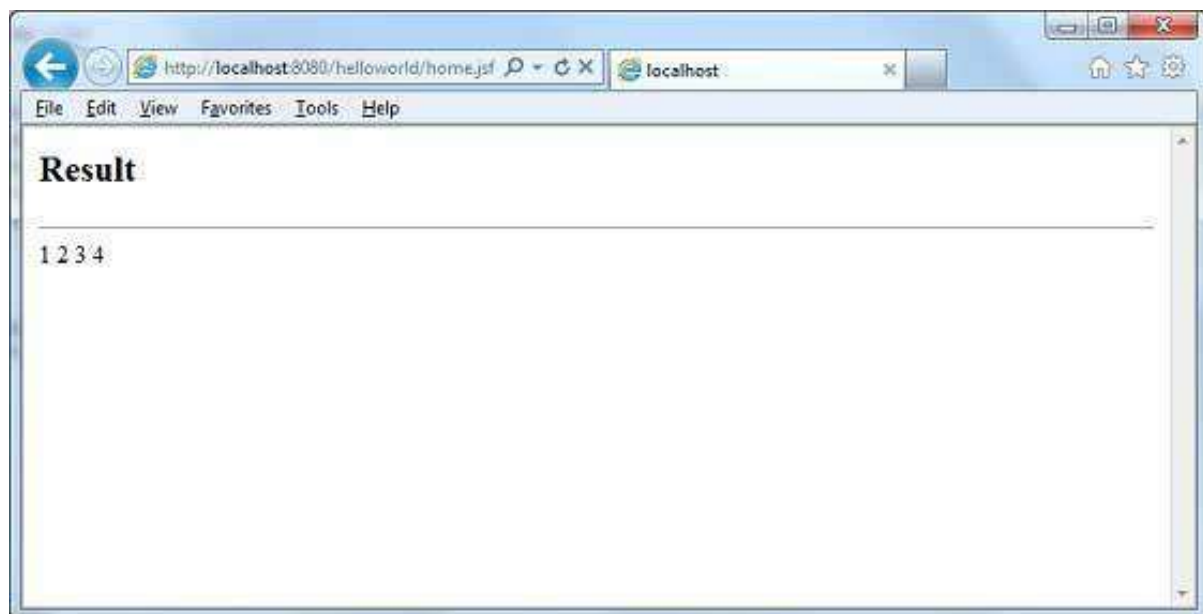
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
    <h:body>
        <h2>Result</h2>
        <hr />
        <ui:repeat value="#{userData.data}" var="s">
            #{s}
        </ui:repeat>
    </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.



Select multiple checkboxes and press **Submit** button. We've selected four items. You will see the selected results.



## h:selectOneRadio

The h:selectOneRadio tag renders a set of HTML input element of type "radio". Format it with HTML table and label tags.

### JSF Tag

```
<h:selectOneRadio value="#{userData.data}">
  <f:selectItem itemValue="1" itemLabel="Item 1" />
  <f:selectItem itemValue="2" itemLabel="Item 2" />
</h:selectOneRadio>
```

## Rendered Output

```
<table>
  <tr>
    <td><input type="radio" checked="checked" name="j_idt6:j_idt8"
      id="j_idt6:j_idt8:0" value="1" />
      <label for="j_idt6:j_idt8:0"> Item 1</label></td>
    <td><input type="radio" name="j_idt6:j_idt8"
      id="j_idt6:j_idt8:1" value="2" />
      <label for="j_idt6:j_idt8:1"> Item 2</label></td>
  </tr>
</table>
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b>

	Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>charset</b> Character encoding for a linked resource
15	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
16	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
17	<b>disabled</b> Disabled state of an input element or button
18	<b>hreflang</b>

	Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
19	<b>lang</b> Base language of an element's attributes and text
20	<b>maxlength</b> Maximum number of characters for text fields
21	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited
22	<b>rel</b> Relationship between the current document and a link specified with the <b>href</b> attribute
23	<b>rev</b> Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types
24	<b>rows</b> Number of visible rows in a text area. <b>h:dataTable</b> has a <b>rows</b> attribute, but it's not an HTML pass-through attribute
25	<b>shape</b> Shape of a region. Valid values: <b>default, rect, circle, poly</b> . (default signifies the entire region)
26	<b>style</b> Inline style information

27	<b>tabindex</b> Numerical value specifying a tab index
28	<b>target</b> The name of a frame in which a document is opened
29	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
30	<b>type</b> Type of a link; for example, <b>stylesheet</b>
31	<b>width</b> Width of an element
32	<b>onblur</b> Element loses focus
33	<b>onchange</b> Element's value changes
34	<b>onclick</b> Mouse button is clicked over the element
35	<b>ondblclick</b> Mouse button is double-clicked over the element
36	<b>onfocus</b> Element receives focus
37	<b>onkeydown</b> Key is pressed
38	<b>onkeypress</b> Key is pressed and subsequently released



39	<b>onkeyup</b> Key is released
40	<b>onmousedown</b> Mouse button is pressed over the element
41	<b>onmousemove</b> Mouse moves over the element
42	<b>onmouseout</b> Mouse leaves the element's area
43	<b>onmouseover</b> Mouse moves onto an element
44	<b>onmouseup</b> Mouse button is released
45	<b>onreset</b> Form is reset
46	<b>onselect</b> Text is selected in an input field
47	<b>disabledClass</b> CSS class for disabled elements
48	<b>enabledClass</b> CSS class for enabled elements
49	<b>layout</b>  Specification for how elements are laid out: <code>lineDirection</code> (horizontal) or <code>pageDirection</code> (vertical)
50	<b>border</b> Border of the element

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```

**home.xhtml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
```

```

<h2>h:selectManyCheckbox example</h2>
<hr />
<h:form>
    <h3>Radio Button</h3>
    <h:selectOneRadio value="#{userData.data}">
        <f:selectItem itemValue="1" itemLabel="Item 1" />
        <f:selectItem itemValue="2" itemLabel="Item 2" />
        <f:selectItem itemValue="3" itemLabel="Item 3" />
        <f:selectItem itemValue="4" itemLabel="Item 4" />
        <f:selectItem itemValue="5" itemLabel="Item 5" />
    </h:selectOneRadio>
    <h:commandButton value="Submit" action="result" />
</h:form>
</h:body>
</html>

```

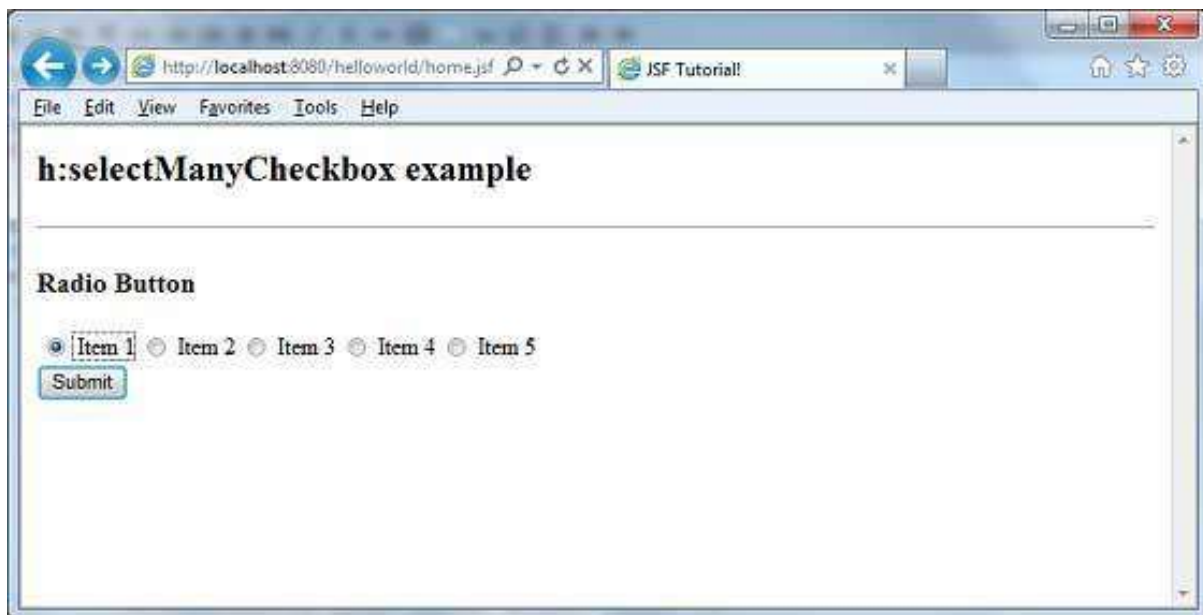
### result.xhtml

```

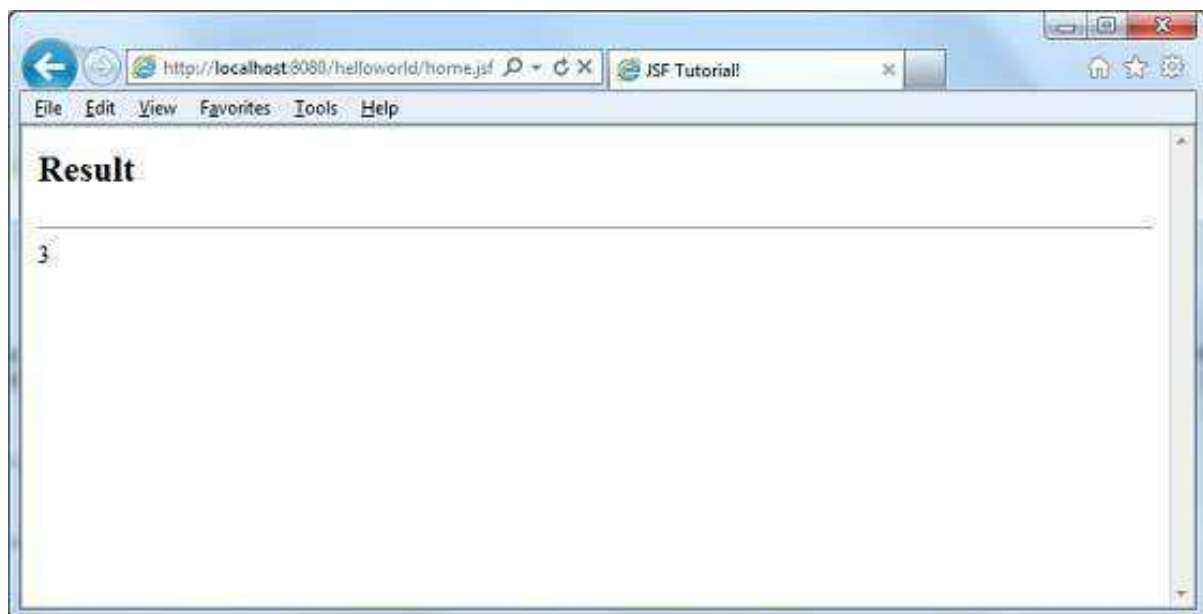
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
        <h2>Result</h2>
        <hr />
        <#{userData.data}>
    </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Select any option and press **Submit** button. We've selected item 3. You will see the selected results.



## h:selectOneListbox

The h:selectOneListbox tag renders an HTML input element of the type "select" with size specified.

## JSF Tag

```
<h:selectOneListbox value="#{userData.data}">
  <f:selectItem itemValue="1" itemLabel="Item 1" />
  <f:selectItem itemValue="2" itemLabel="Item 2" />
</h:selectOneListbox>
```

```
</h:selectOneListbox>
```

## Rendered Output

```
<select name="j_idt6:j_idt8" size="2">
  <option value="1">Item 1</option>
  <option value="2">Item 2</option>
</select>
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	

	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>charset</b> Character encoding for a linked resource
15	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
16	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
17	<b>disabled</b> Disabled state of an input element or button
18	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
19	<b>lang</b>

	Base language of an element's attributes and text
20	<b>maxlength</b> Maximum number of characters for text fields
21	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited
22	<b>rel</b> Relationship between the current document and a link specified with the <b>href</b> attribute
23	<b>rev</b> Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types
24	<b>rows</b> Number of visible rows in a text area. <b>h:dataTable</b> has a <b>rows</b> attribute, but it's not an HTML pass-through attribute
25	<b>shape</b> Shape of a region. Valid values: <b>default</b> , <b>rect</b> , <b>circle</b> , <b>poly</b> (default signifies the entire region)
26	<b>style</b> Inline style information
27	<b>tabindex</b> Numerical value specifying a tab index
28	<b>target</b>



	The name of a frame in which a document is opened
29	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
30	<b>type</b> Type of a link; for example, <b>stylesheet</b>
31	<b>width</b> Width of an element
32	<b>onblur</b> Element loses focus
33	<b>onchange</b> Element's value changes
34	<b>onclick</b> Mouse button is clicked over the element
35	<b>ondblclick</b> Mouse button is double-clicked over the element
36	<b>onfocus</b> Element receives focus
37	<b>onkeydown</b> Key is pressed
38	<b>onkeypress</b> Key is pressed and subsequently released
39	<b>onkeyup</b> Key is released

40	<b>onmousedown</b> Mouse button is pressed over the element
41	<b>onmousemove</b> Mouse moves over the element
42	<b>onmouseout</b> Mouse leaves the element's area
43	<b>onmouseover</b> Mouse moves onto an element
44	<b>onmouseup</b> Mouse button is released
45	<b>onreset</b> Form is reset
46	<b>onselect</b> Text is selected in an input field
47	<b>size</b> Size of input field

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.

2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### UserData.java

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";
}
```

```

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}

```

### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
        <h2>h::selectOneListbox example</h2>
        <hr />
        <h:form>
            <h3>List Box</h3>
            <h:selectOneListbox value="#{userData.data}">
                <f:selectItem itemValue="1" itemLabel="Item 1" />
                <f:selectItem itemValue="2" itemLabel="Item 2" />
                <f:selectItem itemValue="3" itemLabel="Item 3" />
                <f:selectItem itemValue="4" itemLabel="Item 4" />
                <f:selectItem itemValue="5" itemLabel="Item 5" />
            </h:selectOneListbox>
            <h:commandButton value="Submit" action="result" />
        </h:form>
    </h:body>

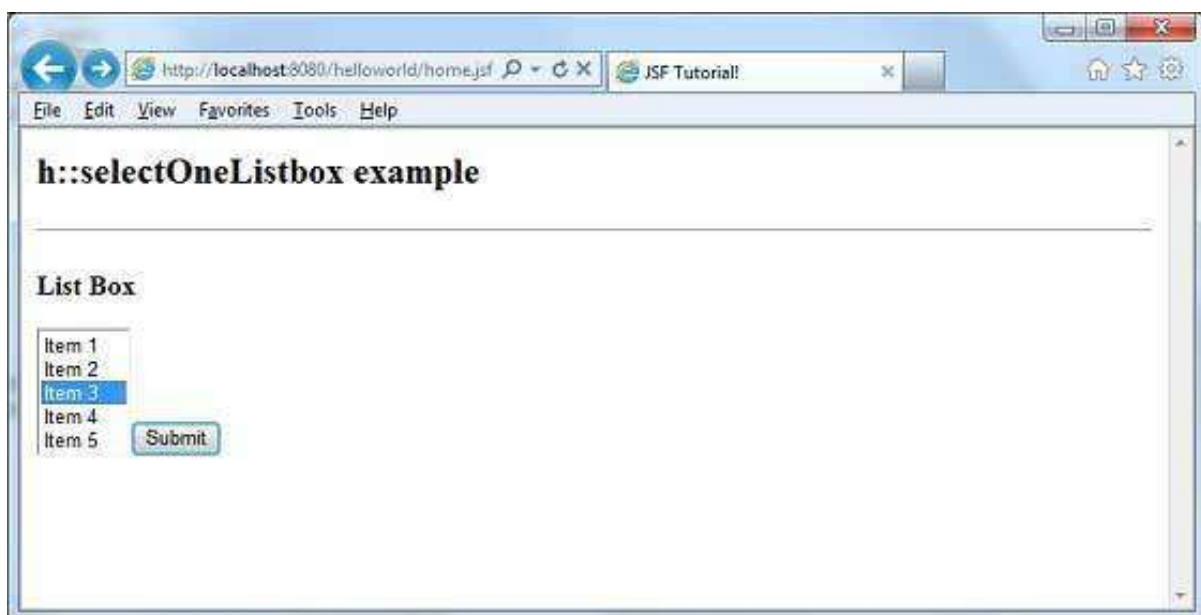
```

```
</html>
```

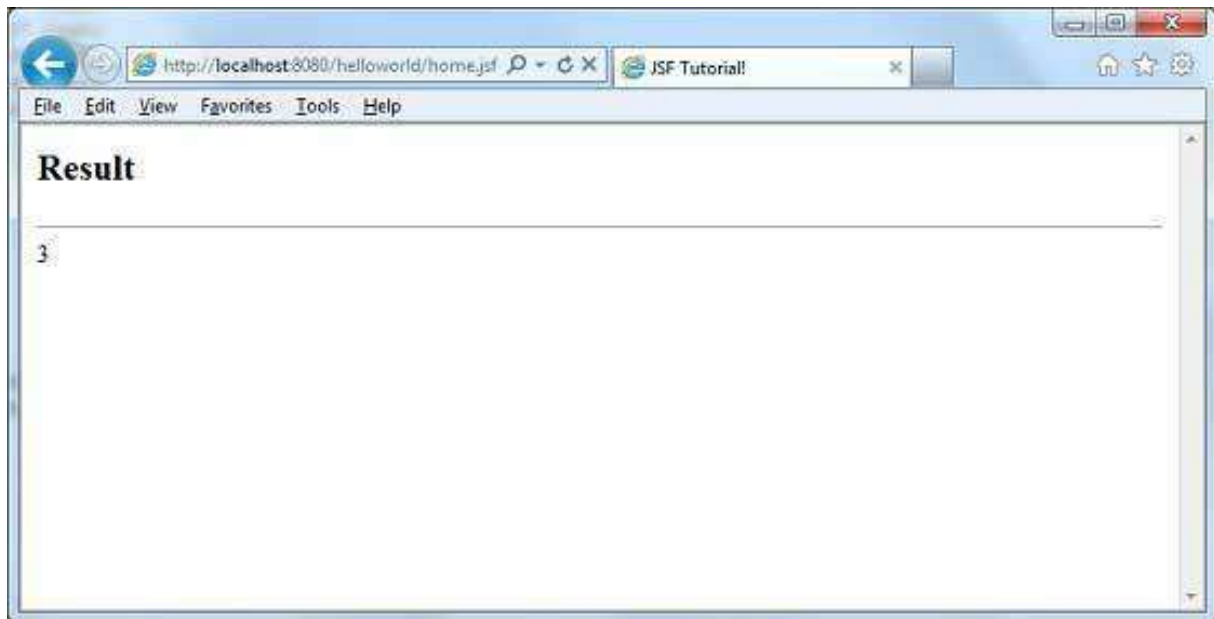
### result.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <head>
    <title>JSF Tutorial!</title>
  </head>
  <h:body>
    <h2>Result</h2>
    <hr />
    #{userData.data}
  </h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Select any option and press **Submit** button. We've selected item 3. You will see the selected results.



## h:selectManyListbox

The `h:selectManyListbox` tag renders an HTML input element of the type "select" with **size** and **multiple** specified.

### JSF Tag

```
<h:selectManyListbox value="#{userData.data}">
    <f:selectItem itemValue="1" itemLabel="Item 1" />
    <f:selectItem itemValue="2" itemLabel="Item 2" />
</h:selectManyListbox>
```

### Rendered Output

```
<select name="j_idt6:j_idt8" size="2" multiple="multiple">
    <option value="1">Item 1</option>
    <option value="2">Item 2</option>
</select>
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element

11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>charset</b> Character encoding for a linked resource
15	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
16	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
17	<b>disabled</b> Disabled state of an input element or button
18	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
19	<b>lang</b> Base language of an element's attributes and text
20	<b>maxlength</b> Maximum number of characters for text fields
21	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited
22	



	<p><b>rel</b></p> <p>Relationship between the current document and a link specified with the <b>href</b> attribute</p>
23	<p><b>rev</b></p> <p>Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types</p>
24	<p><b>rows</b></p> <p>Number of visible rows in a text area. <b>h:dataTable</b> has a <b>rows</b> attribute, but it's not an HTML pass-through attribute</p>
25	<p><b>shape</b></p> <p>Shape of a region. Valid values: <b>default, rect, circle, poly</b>. (default signifies the entire region)</p>
26	<p><b>style</b></p> <p>Inline style information</p>
27	<p><b>tabindex</b></p> <p>Numerical value specifying a tab index</p>
28	<p><b>target</b></p> <p>The name of a frame in which a document is opened</p>
29	<p><b>title</b></p> <p>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value</p>
30	<p><b>type</b></p> <p>Type of a link; for example, <b>stylesheet</b></p>

31	<b>width</b> Width of an element
32	<b>onblur</b> Element loses focus
33	<b>onchange</b> Element's value changes
34	<b>onclick</b> Mouse button is clicked over the element
35	<b>ondblclick</b> Mouse button is double-clicked over the element
36	<b>onfocus</b> Element receives focus
37	<b>onkeydown</b> Key is pressed
38	<b>onkeypress</b> Key is pressed and subsequently released
39	<b>onkeyup</b> Key is released
40	<b>onmousedown</b> Mouse button is pressed over the element
41	<b>onmousemove</b> Mouse moves over the element
42	<b>onmouseout</b> Mouse leaves the element's area
43	<b>onmouseover</b>

	Mouse moves onto an element
44	<b>onmouseup</b> Mouse button is released
45	<b>onreset</b> Form is reset
46	<b>onselect</b> Text is selected in an input field
47	<b>size</b> Size of input field

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure business logic is working as per the requirements.

6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### UserData.java

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String[] data = {"1","2","3"};

    public String[] getData() {
        return data;
    }

    public void setData(String[] data) {
        this.data = data;
    }

}
```

### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
        <h2>h:selectManyListbox example</h2>
        <hr />
        <h:form>
            <h3>List Box</h3>
            <h:selectManyListbox value="#{userData.data}">
                <f:selectItem itemValue="1" itemLabel="Item 1" />
                <f:selectItem itemValue="2" itemLabel="Item 2" />
                <f:selectItem itemValue="3" itemLabel="Item 3" />
                <f:selectItem itemValue="4" itemLabel="Item 4" />
                <f:selectItem itemValue="5" itemLabel="Item 5" />
            </h:selectManyListbox>
            <h:commandButton value="Submit" action="result" />
        </h:form>
    </h:body>
</html>

```

### result.xhtml

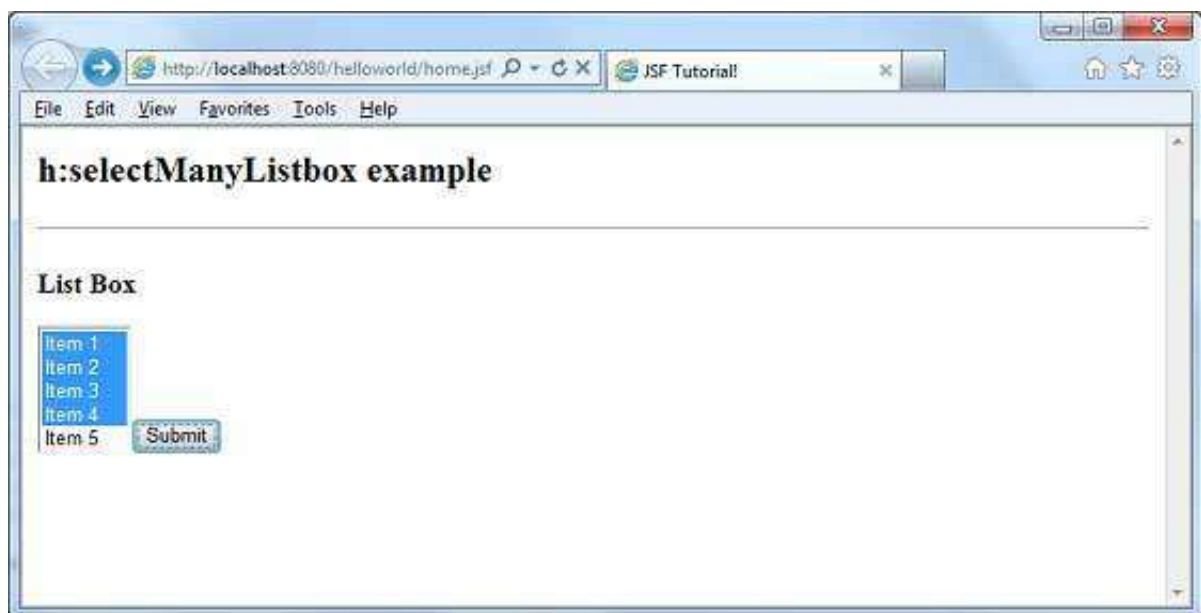
```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
    <h:body>
        <h2>Result</h2>
    </h:body>

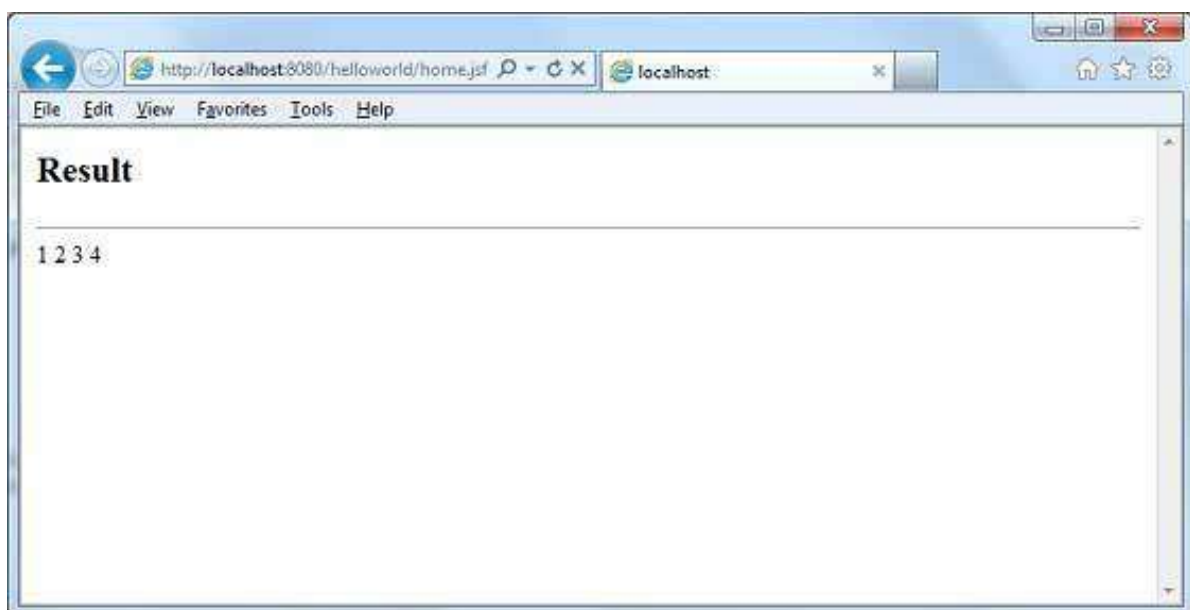
```

```
<hr />
<ui:repeat value="#{userData.data}" var="s">
    #{s}
</ui:repeat>
</h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.



Select multiple values and press **Submit** button. We've selected four items. You will see the selected results.



## h:selectOneMenu

The h:selectOneMenu tag renders an HTML input element of the type "select" with **size** not specified.

### JSF Tag

```
<h:selectOneMenu value="#{userData.data}">
    <f:selectItem itemValue="1" itemLabel="Item 1" />
    <f:selectItem itemValue="2" itemLabel="Item 2" />
</h:selectOneMenu>
```

### Rendered Output

```
<select name="j_idt6:j_idt8">
    <option value="1">Item 1</option>
    <option value="2">Item 2</option>
</select>
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name

5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>charset</b> Character encoding for a linked resource



15	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
16	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
17	<b>disabled</b> Disabled state of an input element or button
18	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
19	<b>lang</b> Base language of an element's attributes and text
20	<b>maxlength</b> Maximum number of characters for text fields
21	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited
22	<b>rel</b> Relationship between the current document and a link specified with the <b>href</b> attribute
23	<b>rev</b> Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types

24	<b>rows</b> Number of visible rows in a text area. <b>h:dataTable</b> has a <b>rows</b> attribute, but it's not an HTML pass-through attribute
25	<b>shape</b> Shape of a region. Valid values: <b>default</b> , <b>rect</b> , <b>circle</b> , <b>poly</b> . (default signifies the entire region)
26	<b>style</b> Inline style information
27	<b>tabindex</b> Numerical value specifying a tab index
28	<b>target</b> The name of a frame in which a document is opened
29	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
30	<b>type</b> Type of a link; for example, <b>stylesheet</b>
31	<b>width</b> Width of an element
32	<b>onblur</b> Element loses focus
33	<b>onchange</b> Element's value changes

34	<b>onclick</b> Mouse button is clicked over the element
35	<b>ondblclick</b> Mouse button is double-clicked over the element
36	<b>onfocus</b> Element receives focus
37	<b>onkeydown</b> Key is pressed
38	<b>onkeypress</b> Key is pressed and subsequently released
39	<b>onkeyup</b> Key is released
40	<b>onmousedown</b> Mouse button is pressed over the element
41	<b>onmousemove</b> Mouse moves over the element
42	<b>onmouseout</b> Mouse leaves the element's area
43	<b>onmouseover</b> Mouse moves onto an element
44	<b>onmouseup</b> Mouse button is released
45	<b>onreset</b> Form is reset

46	<b>onselect</b> Text is selected in an input field
----	---

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

}
```

**home.xhtml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
```

```

<h2>h::selectOneMenu example</h2>
<hr />
<h:form>
    <h3>Combo Box</h3>
    <h:selectOneMenu value="#{userData.data}">
        <f:selectItem itemValue="1" itemLabel="Item 1" />
        <f:selectItem itemValue="2" itemLabel="Item 2" />
        <f:selectItem itemValue="3" itemLabel="Item 3" />
        <f:selectItem itemValue="4" itemLabel="Item 4" />
        <f:selectItem itemValue="5" itemLabel="Item 5" />

    </h:selectOneMenu>
    <h:commandButton value="Submit" action="result" />
</h:form>
</h:body>
</html>

```

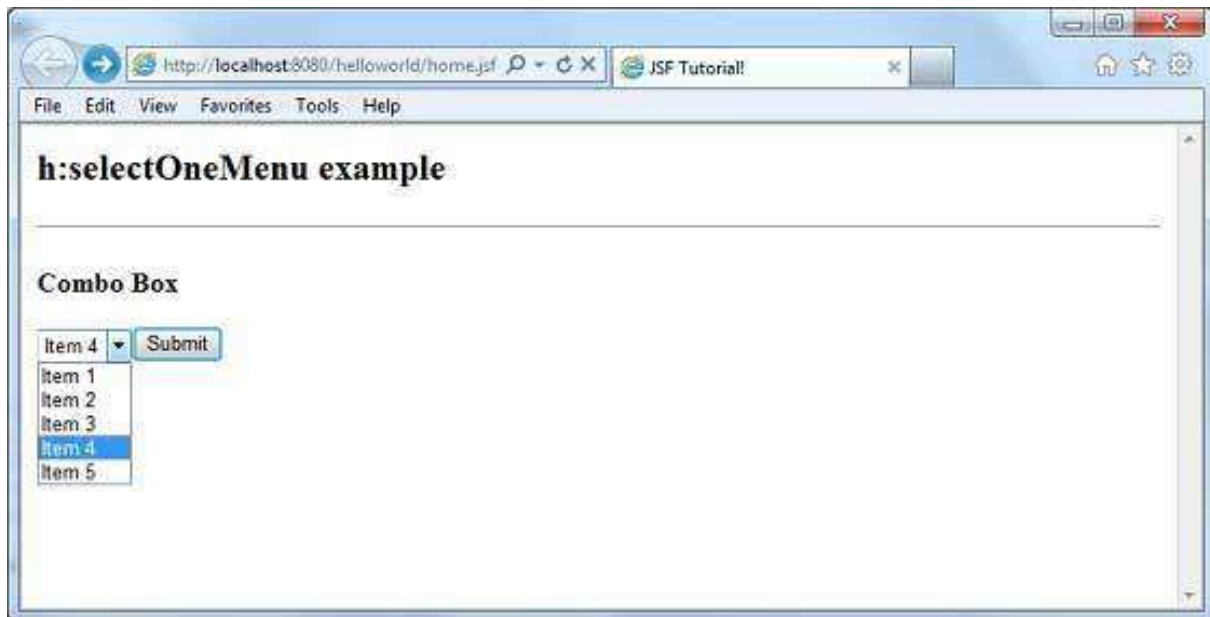
### result.xhtml

```

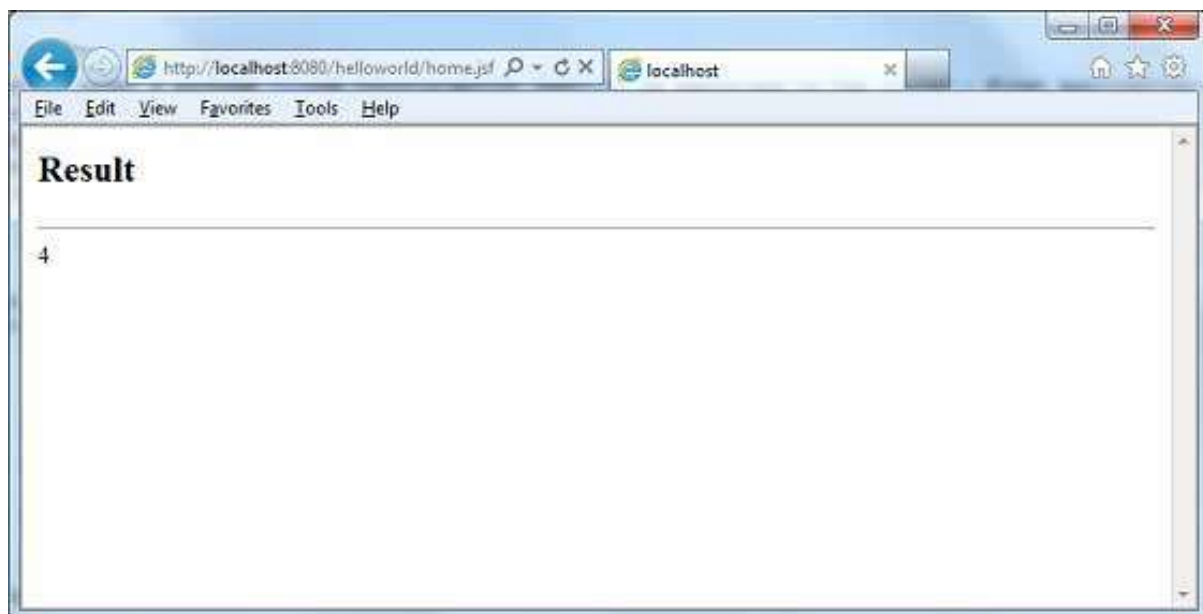
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
        <h2>Result</h2>
        <hr />
        <div>
            #{userData.data}
        </div>
    </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Select any option and press **Submit** button. We've selected item 4. You will see the selected results.



## h:outputText

The h:outputText tag renders an HTML text.

### JSF Tag

```
<h:outputText value="Hello World!" />
```

### Rendered Output

Hello World!

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>converter</b> Converter class name
7	<b>style</b> Inline style information



8	<p><b>title</b></p> <p>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value</p>
---	--

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

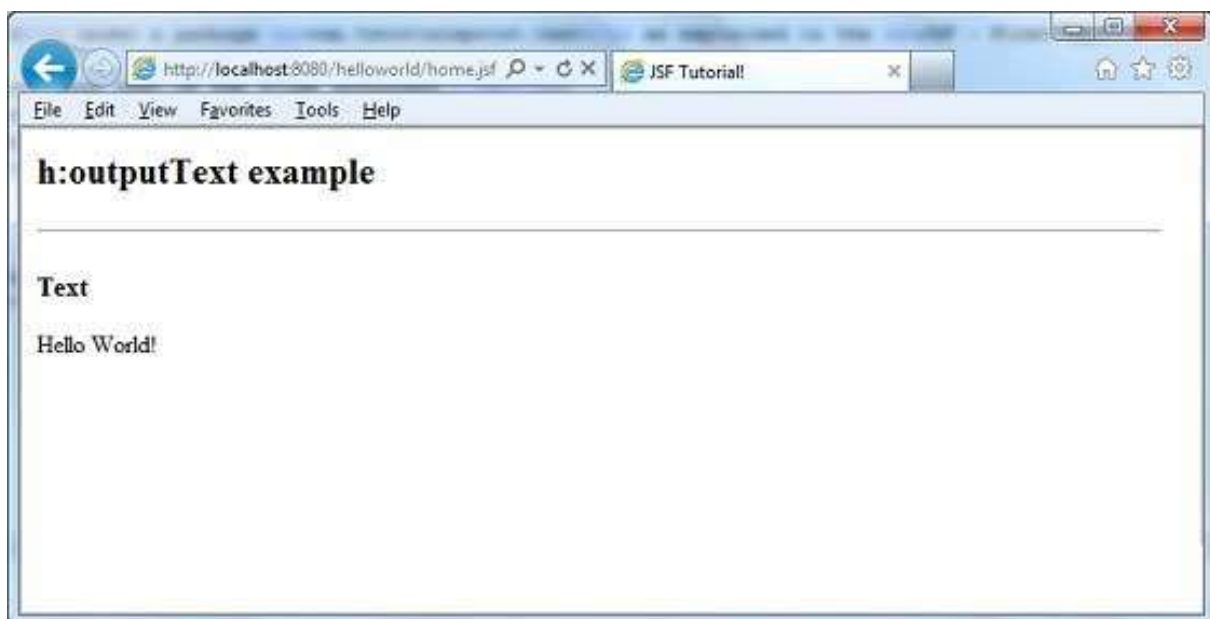
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>JSF Tutorial!</title>
```

```

</head>
<body>
  <h2>h:outputText example</h2>
  <hr />
  <h:form>
    <h3>Text</h3>
    <h:outputText value="Hello World"/>
  </h:form>
</body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.



## h:outputFormat

The h:outputFormat tag renders an HTML text but can accept parameterised inputs.

### JSF Tag

```

<h:outputFormat value="parameter 1 : {0}, parameter 2 : {1}" >
  <f:param value="Item 1" />
  <f:param value="Item 2" />
</h:outputFormat>

```

## Rendered Output

parameter 1 : Item 1, parameter 2 : Item 2

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>converter</b> Converter class name
7	<b>style</b> Inline style information
8	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

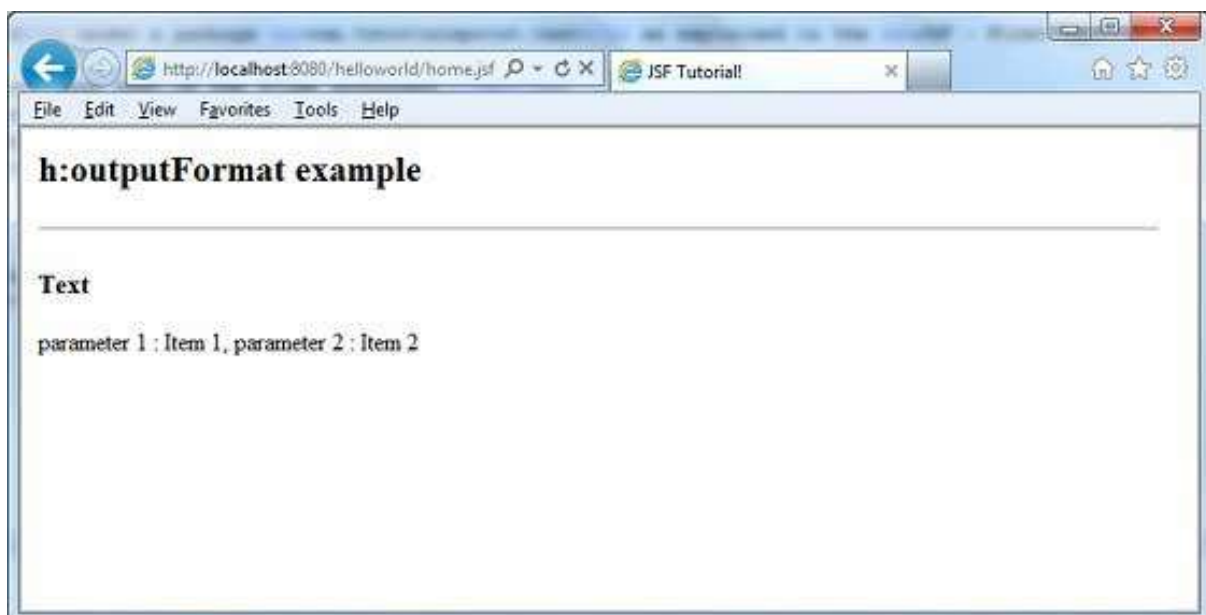
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:outputFormat example</h2>
    <hr />
    <h:form>
        <h3>Text</h3>
```

```

    <h:outputFormat value="parameter 1 : {0}, parameter 2 : {1}" >
        <f:param value="Item 1" />
        <f:param value="Item 2" />
    </h:outputFormat>
</h:form>
</body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:graphicImage

The h:graphicImage tag renders an HTML element of the type "img".

### JSF Tag

```

<h:graphicImage
    value="http://www.tutorialspoint.com/images/jsf-mini-logo.png"/>

```

### Rendered Output

```



```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>alt</b> Alternative text for nontextual elements such as images or applets
7	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
8	<b>lang</b> Base language of an element's attributes and text
9	<b>style</b> Inline style information
10	<b>tabindex</b> Numerical value specifying a tab index
11	<b>target</b> The name of a frame in which a document is opened

12	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
13	<b>usemap</b> Usemap of an element
14	<b>url</b> Url of the image
15	<b>width</b> Width of an element
16	<b>onblur</b> Element loses focus
17	<b>onchange</b> Element's value changes
18	<b>onclick</b> Mouse button is clicked over the element
19	<b>ondblclick</b> Mouse button is double-clicked over the element
20	<b>onfocus</b> Element receives focus
21	<b>onkeydown</b> Key is pressed
22	<b>onkeypress</b> Key is pressed and subsequently released
23	<b>onkeyup</b> Key is released

24	<b>onmousedown</b> Mouse button is pressed over the element
25	<b>onmousemove</b> Mouse moves over the element
26	<b>onmouseout</b> Mouse leaves the element's area
27	<b>onmouseover</b> Mouse moves onto an element
28	<b>onmouseup</b> Mouse button is released

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
```



```

</head>
<body>
    <h2>h:graphicImage example</h2>
    <hr />
    <h:form>
        <h3>Image</h3>
        <h:graphicImage value="/images/jsf-mini-logo.png"/>
    </h:form>
</body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:outputStylesheet

The `h:outputStylesheet` tag renders an HTML element of the type "link" with type "text/css". This tag is used to add external stylesheet file to JSF page.

## JSF Tag

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

## Rendered Output

```
<link type="text/css" rel="stylesheet"
      href="/helloworld/javax.faces.resource/styles.css.jsf?ln=css" />
```

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Create <i>resources</i> folder under <i>src &gt; main</i> folder.
3	Create <i>css</i> folder under <i>src &gt; main &gt; resources</i> folder.
4	Create <i>styles.css</i> file under <i>src &gt; main &gt; resources &gt; css</i> folder.
5	Modify <i>styles.css</i> file as explained below.
6	Modify <i>pom.xml</i> as explained below.
7	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
8	Compile and run the application to make sure business logic is working as per the requirements.
9	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

10	Launch your web application using appropriate URL as explained below in the last step.
----	--

**styles.css**

```
.message{
    color:green;
}
```

**pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tutorialspoint.test</groupId>
  <artifactId>helloworld</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-api</artifactId>
      <version>2.1.7</version>
    </dependency>
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-impl</artifactId>
      <version>2.1.7</version>
    </dependency>
  </dependencies>
</project>
```

```

</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
</dependencies>
<build>
  <finalName>helloworld</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.1</version>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-resources-plugin</artifactId>
      <version>2.6</version>
      <executions>
        <execution>
          <id>copy-resources</id>
          <phase>validate</phase>
          <goals>
            <goal>copy-resources</goal>
          </goals>
          <configuration>
            <outputDirectory>${basedir}/target/helloworld/resources
            </outputDirectory>
            <resources>
              <resource>
                <directory>src/main/resources</directory>
                <filtering>true</filtering>
              </resource>
            </resources>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

```

        </execution>
    </executions>
</plugin>
</plugins>
</build>
</project>

```

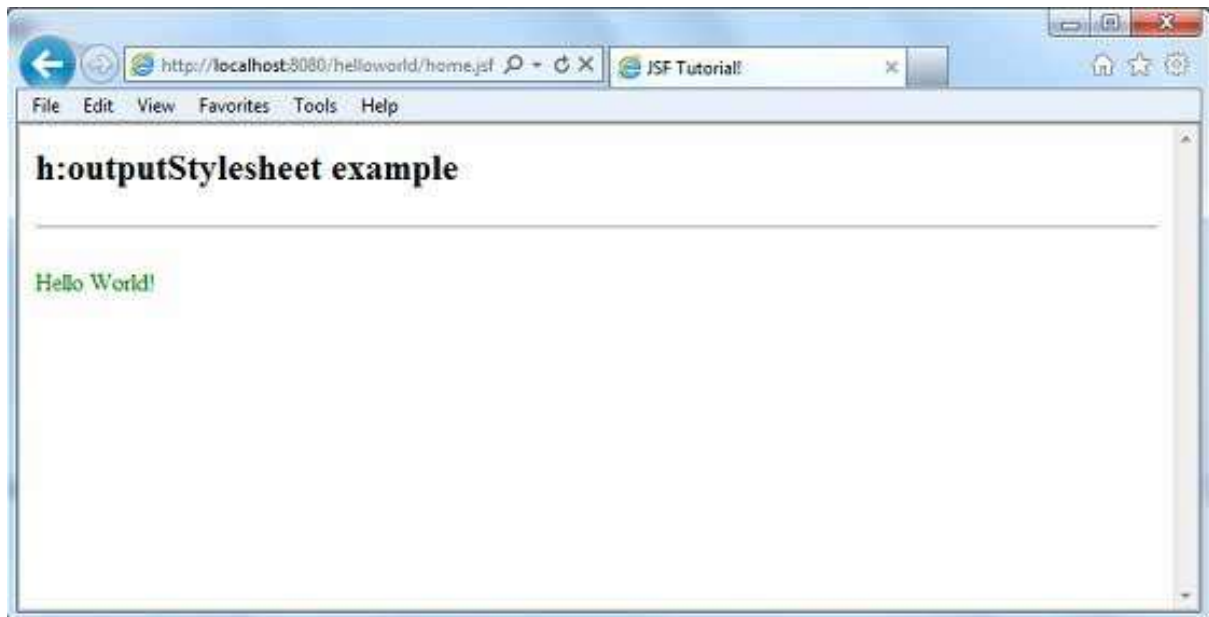
## home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>JSF Tutorial!</title>
        <h:outputStylesheet library="css" name="styles.css" />
    </h:head>
    <h:body>
        <h2>h:outputStylesheet example</h2>
        <hr />
        <h:form>
            <div class="message">Hello World!</div>
        </h:form>
    </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:outputScript

The `h:outputScript` tag renders an HTML element of the type "script" with type "text/javascript". This tag is used to add an external javascript file to JSF page.

## JSF Tag

```
<h:outputScript library="js" name="help.js" />
```

## Rendered Output

```
<script type="text/javascript"
  src="/helloworld/javax.faces.resource/help.js.jsf?ln=js"></script>
```

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Create <i>resources</i> folder under <i>src &gt; main</i> folder.
3	Create <i>js</i> folder under <i>src &gt; main &gt; resources</i> folder.
4	Create <i>help.js</i> file under <i>src &gt; main &gt; resources &gt; js</i> folder.
5	Modify <i>help.js</i> file as explained below.
6	Modify <i>pom.xml</i> as explained below.
7	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
8	Compile and run the application to make sure business logic is working as per the requirements.

9	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
10	Launch your web application using appropriate URL as explained below in the last step.

### help.js

```
function showMessage(){
    alert("Hello World!");
}
```

### pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tutorialspoint.test</groupId>
  <artifactId>helloworld</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-api</artifactId>
```



```

        <version>2.1.7</version>
    </dependency>
    <dependency>
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-impl</artifactId>
        <version>2.1.7</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
</dependencies>
<build>
    <finalName>helloworld</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.1</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-resources-plugin</artifactId>
            <version>2.6</version>
            <executions>
                <execution>
                    <id>copy-resources</id>
                    <phase>validate</phase>
                    <goals>
                        <goal>copy-resources</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>${basedir}/target/helloworld/resources

```

```

        </outputDirectory>
    </resources>
    <resource>
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
    </resource>
</resources>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>

```

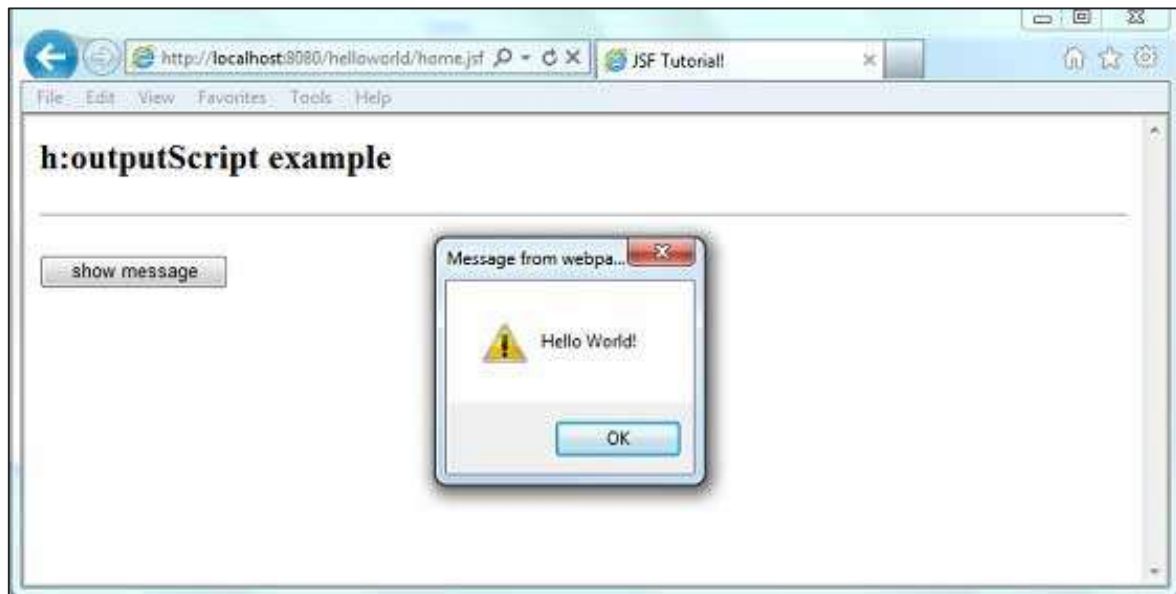
## home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>JSF Tutorial!</title>
        <h:outputScript library="js" name="help.js" />
    </h:head>
    <h:body>
        <h2>h:outputScript example</h2>
        <hr />
        <h:form>
            <h:commandButton onclick="showMessage();" />
        </h:form>
    </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:commandButton

The h:commandButton tag renders an HTML input element of the type "submit".

## JSF Tag

```
<h:commandButton value="Click Me!" onclick="alert('Hello World!');" />
```

## Rendered Output

```
<input type="submit" name="j_idt10:j_idt13" value="Click Me!"  
  onclick="alert('Hello World!');" />
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>rendered</b> A boolean; false suppresses rendering
3	<b>value</b> A component's value, typically a value binding
4	<b>valueChangeListener</b> A method binding to a method that responds to value changes
5	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
6	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
7	<b>disabled</b> Disabled state of an input element or button
8	<b>tabindex</b> Numerical value specifying a tab index
9	<b>target</b> The name of a frame in which a document is opened
10	<b>title</b>  A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
11	<b>width</b>

	Width of an element
12	<b>onblur</b> Element loses focus
13	<b>onchange</b> Element's value changes
14	<b>onclick</b> Mouse button is clicked over the element
15	<b>ondblclick</b> Mouse button is double-clicked over the element
16	<b>onfocus</b> Element receives focus
17	<b>onkeydown</b> Key is pressed
18	<b>onkeypress</b> Key is pressed and subsequently released
19	<b>onkeyup</b> Key is released
20	<b>onmousedown</b> Mouse button is pressed over the element
21	<b>onmousemove</b> Mouse moves over the element
22	<b>onmouseout</b> Mouse leaves the element's area

23	<b>onmouseover</b> Mouse moves onto an element
24	<b>onmouseup</b> Mouse button is released
25	<b>onreset</b> Form is reset
26	<b>onselect</b> Text is selected in an input field

## Example Application

Let us create a test JSF application to test the above tag.

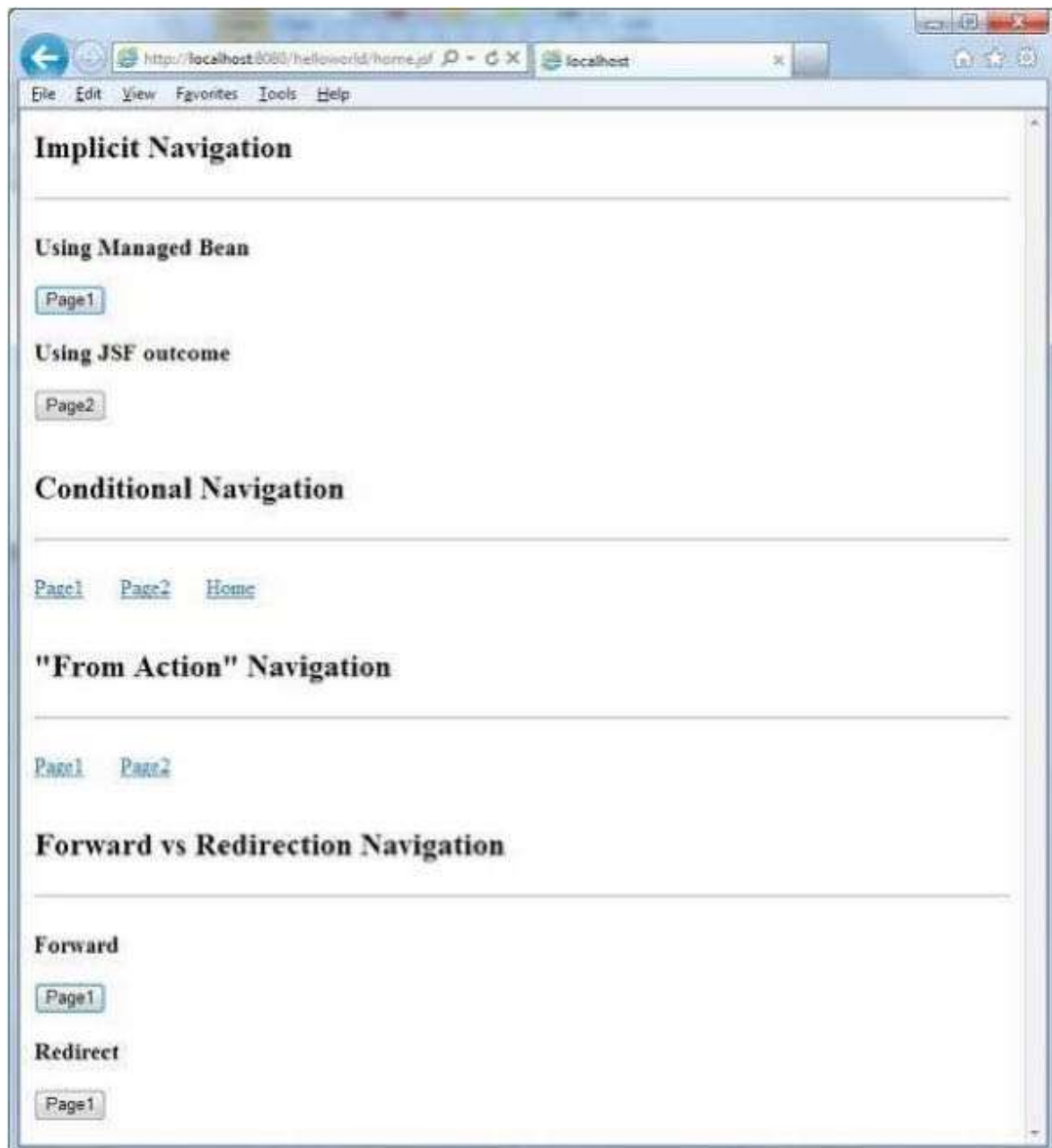
Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

5	Launch your web application using appropriate URL as explained below in the last step.
---	--

**home.xhtml**

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:commandButton example</h2>
    <hr />
    <h:form>
        <h:commandButton value="Click Me!" onclick="alert('Hello World!');" />
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:Link

The h:Link tag renders an HTML "anchor" element.

## JSF Tag

```
<h:link value="Page 1" outcome="page1" />
```



## Rendered Output

```
<a href="/helloworld/page1.jsf">Page 1</a>
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form

12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>border</b> Pixel value for an element's border width
15	<b>charset</b> Character encoding for a linked resource
16	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
17	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
18	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
19	<b>lang</b> Base language of an element's attributes and text
20	<b>maxlength</b> Maximum number of characters for text fields
21	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited
22	<b>rel</b> Relationship between the current document and a link specified with the <b>href</b> attribute
23	<b>rev</b>  Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types

24	<b>size</b> Size of an input field
25	<b>style</b> Inline style information
26	<b>tabindex</b> Numerical value specifying a tab index
27	<b>target</b> The name of a frame in which a document is opened
28	<b>title</b>  A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
29	<b>type</b> Type of a link; for example, <b>stylesheet</b>
30	<b>width</b> Width of an element
31	<b>onblur</b> Element loses focus
32	<b>onchange</b> Element's value changes
33	<b>onclick</b> Mouse button is clicked over the element
34	<b>ondblclick</b> Mouse button is double-clicked over the element
35	<b>onfocus</b> Element receives focus
36	

	<b>onkeydown</b> Key is pressed
37	<b>onkeypress</b> Key is pressed and subsequently released
38	<b>onkeyup</b> Key is released
39	<b>onmousedown</b> Mouse button is pressed over the element
40	<b>onmousemove</b> Mouse moves over the element
41	<b>onmouseout</b> Mouse leaves the element's area
42	<b>onmouseover</b> Mouse moves onto an element
43	<b>onmouseup</b> Mouse button is released
44	<b>onreset</b> Form is reset
45	<b>onselect</b> Text is selected in an input field

## Example Application

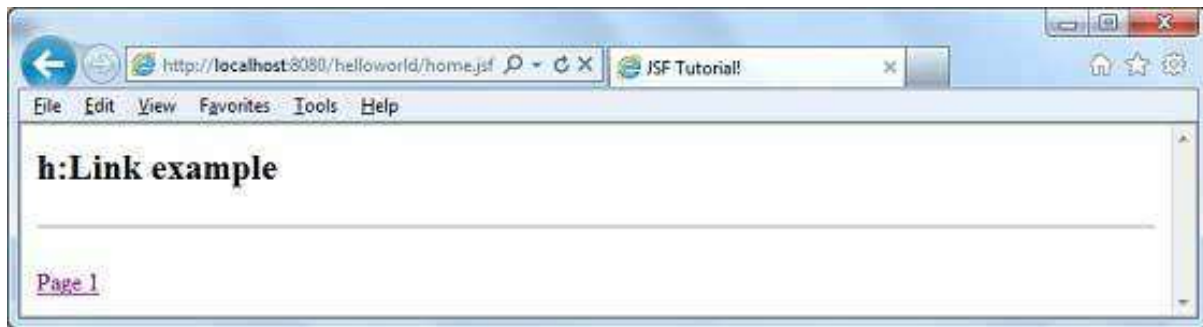
Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:Link example</h2>
    <hr />
    <h:form>
        <h:link value="Page 1" outcome="page1" />
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:commandLink

The h:commandLink tag renders an HTML "anchor" element.

### JSF Tag

```
<h:commandLink value="Page 1" action="page1" />
```

### Rendered Output

```
<a href="#" onclick="mojarra.jsfcljs(document.getElementById('j_idt13'),
{'j_idt13:j_idt14':'j_idt13:j_idt14'}, '');return false">Page 1</a>
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name

5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>Accesskey</b> A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form
12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>Alt</b> Alternative text for nontextual elements such as images or applets
14	<b>border</b> Pixel value for an element's border width
15	<b>charset</b> Character encoding for a linked resource

16	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
17	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
18	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
19	<b>lang</b> Base language of an element's attributes and text
20	<b>maxlength</b> Maximum number of characters for text fields
21	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited
22	<b>rel</b> Relationship between the current document and a link specified with the <b>href</b> attribute
23	<b>rev</b> Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types
24	<b>size</b> Size of an input field
25	<b>style</b> Inline style information



26	<b>tabindex</b> Numerical value specifying a tab index
27	<b>target</b> The name of a frame in which a document is opened
28	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
29	<b>type</b> Type of a link; for example, <b>stylesheet</b>
30	<b>width</b> Width of an element
31	<b>onblur</b> Element loses focus
32	<b>onchange</b> Element's value changes
33	<b>onclick</b> Mouse button is clicked over the element
34	<b>ondblclick</b> Mouse button is double-clicked over the element
35	<b>onfocus</b> Element receives focus

36	<b>onkeydown</b> Key is pressed
37	<b>onkeypress</b> Key is pressed and subsequently released
38	<b>onkeyup</b> Key is released
39	<b>onmousedown</b> Mouse button is pressed over the element
40	<b>onmousemove</b> Mouse moves over the element
41	<b>onmouseout</b> Mouse leaves the element's area
42	<b>onmouseover</b> Mouse moves onto an element
43	<b>onmouseup</b> Mouse button is released
44	<b>onreset</b> Form is reset
45	<b>onselect</b> Text is selected in an input field

## Example Application

Let us create a test JSF application to test the above tag.

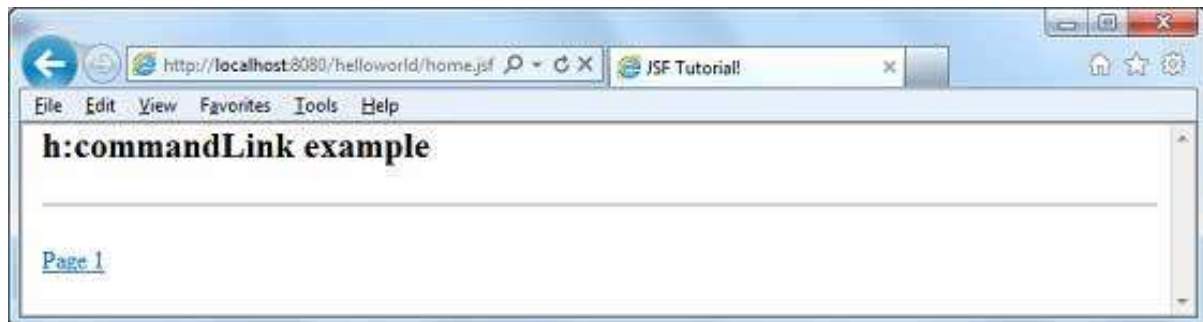
Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:commandLink example</h2>
    <hr />
    <h:form>
        <h:commandLink value="Page 1" action="page1" />
    </h:form>
```

```
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:outputLink

The h:outputLink tag renders an HTML "anchor" element.

### JSF Tag

```
<h:outputLink value="page1.jsf" >Page 1</h:outputLink>
```

### Rendered Output

```
<a href="page1.jsf">Page 1</a>
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component

2	<b>binding</b>
---	----------------

	Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>valueChangeListener</b> A method binding to a method that responds to value changes
7	<b>converter</b> Converter class name
8	<b>validator</b> Class name of a validator that's created and attached to a component
9	<b>required</b> A boolean; if true, requires a value to be entered in the associated field
10	<b>accesskey</b>  A key, typically combined with a system-defined metakey, that gives focus to an element
11	<b>accept</b> Comma-separated list of content types for a form

12	<b>accept-charset</b> Comma- or space-separated list of character encodings for a form. The <b>accept-charset</b> attribute is specified with the JSF HTML attribute named <b>acceptcharset</b>
13	<b>alt</b> Alternative text for nontextual elements such as images or applets
14	<b>border</b> Pixel value for an element's border width
15	<b>charset</b> Character encoding for a linked resource
16	<b>coords</b> Coordinates for an element whose shape is a rectangle, circle, or polygon
17	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
18	<b>hreflang</b> Base language of a resource specified with the <b>href</b> attribute; <b>hreflang</b> may only be used with <b>href</b>
19	<b>lang</b> Base language of an element's attributes and text
20	<b>maxlength</b> Maximum number of characters for text fields
21	<b>readonly</b> Read-only state of an input field; text can be selected in a readonly field but not edited

22	<b>rel</b> Relationship between the current document and a link specified with the <b>href</b> attribute
23	<b>rev</b> Reverse link from the anchor specified with <b>href</b> to the current document. The value of the attribute is a space-separated list of link types
24	<b>size</b> Size of an input field
25	<b>style</b> Inline style information
26	<b>tabindex</b> Numerical value specifying a tab index
27	<b>target</b> The name of a frame in which a document is opened
28	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
29	<b>type</b> Type of a link; for example, <b>stylesheet</b>
30	<b>width</b> Width of an element
31	<b>onblur</b> Element loses focus

32	<b>onchange</b> Element's value changes
33	<b>onclick</b> Mouse button is clicked over the element
34	<b>ondblclick</b> Mouse button is double-clicked over the element
35	<b>onfocus</b> Element receives focus
36	<b>onkeydown</b> Key is pressed
37	<b>onkeypress</b> Key is pressed and subsequently released
38	<b>onkeyup</b> Key is released
39	<b>onmousedown</b> Mouse button is pressed over the element
40	<b>onmousemove</b> Mouse moves over the element
41	<b>onmouseout</b> Mouse leaves the element's area
42	<b>onmouseover</b> Mouse moves onto an element
43	<b>onmouseup</b> Mouse button is released



44	<b>onreset</b> Form is reset
45	<b>onselect</b> Text is selected in an input field

## Example Application

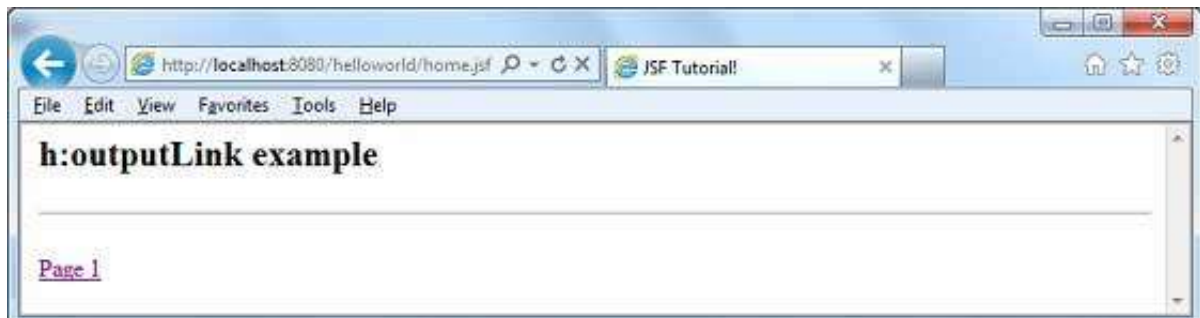
Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:outputLink example</h2>
    <hr />
    <h:form>
        <h:outputLink value="page1.jsf" >Page 1</h:outputLink>
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:panelGrid

The h:panel tag renders an HTML "table" element.

### JSF Tag

```
<h:panelGrid id="panel" columns="2" border="1"
  cellpadding="10" cellspacing="1">
  <f:facet name="header">
    <h:outputText value="Login"/>
  </f:facet>
  <h:outputLabel value="Username" />
  <h:inputText />
  <h:outputLabel value="Password" />
  <h:inputSecret />
  <f:facet name="footer">
    <h:panelGroup style="display:block; text-align:center">
      <h:commandButton id="submit" value="Submit" />
    </h:panelGroup>
  </f:facet>
</h:panelGrid>
```

### Rendered Output

```
<table id="j_idt10:panel" border="1" cellpadding="10" cellspacing="1">
<thead>
  <tr><th colspan="2" scope="colgroup">Login</th></tr>
</thead>
<tfoot>
  <tr>
```

```

        <td colspan="2">
        <span style="display:block; text-align:center">
        <input id="j_idt10:submit" type="submit"
        name="j_idt10:submit" value="Submit" />
        </span></td></tr>
</tfoot>
<tbody>
    <tr>
        <td><label>Username</label></td>
        <td><input type="text" name="j_idt10:j_idt17" /></td>
    </tr>
    <tr>
        <td><label>Password</label></td>
        <td><input type="password" name="j_idt10:j_idt21" value="" /></td>
    </tr>
</tbody>
</table>

```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding
6	<b>bgcolor</b> Background color for the table

7	<b>border</b> Width of the table's border
8	<b>cellpadding</b> Padding around table cells
9	<b>cellspacing</b> Spacing between table cells
10	<b>columnClasses</b> Comma-separated list of CSS classes for columns
11	<b>columns</b> Number of columns in the table
12	<b>footerClass</b> CSS class for the table footer
13	<b>frame</b>  frame Specification for sides of the frame surrounding the table that are to be drawn; valid values: none, above, below, hside, vside, lhs, rhs, box, border
14	<b>headerClass</b> CSS class for the table header
15	<b>rowClasses</b> Comma-separated list of CSS classes for columns
16	<b>rules</b>  Specification for lines drawn between cells; valid values: groups, rows, columns, all
17	<b>summary</b>  Summary of the table's purpose and structure used for non-visual feedback such as speech

18	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
19	<b>lang</b> Base language of an element's attributes and text
20	<b>border</b> Pixel value for an element's border width
21	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
22	<b>width</b> Width of an element
23	<b>onblur</b> Element loses focus
24	<b>onchange</b> Element's value changes
25	<b>onclick</b> Mouse button is clicked over the element
26	<b>ondblclick</b> Mouse button is double-clicked over the element
27	<b>onfocus</b> Element receives focus

28	<b>onkeydown</b> Key is pressed
29	<b>onkeypress</b> Key is pressed and subsequently released
30	<b>onkeyup</b> Key is released
31	<b>onmousedown</b> Mouse button is pressed over the element
32	<b>onmousemove</b> Mouse moves over the element
33	<b>onmouseout</b> Mouse leaves the element's area
34	<b>onmouseover</b> Mouse moves onto an element
35	<b>onmouseup</b> Mouse button is released

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

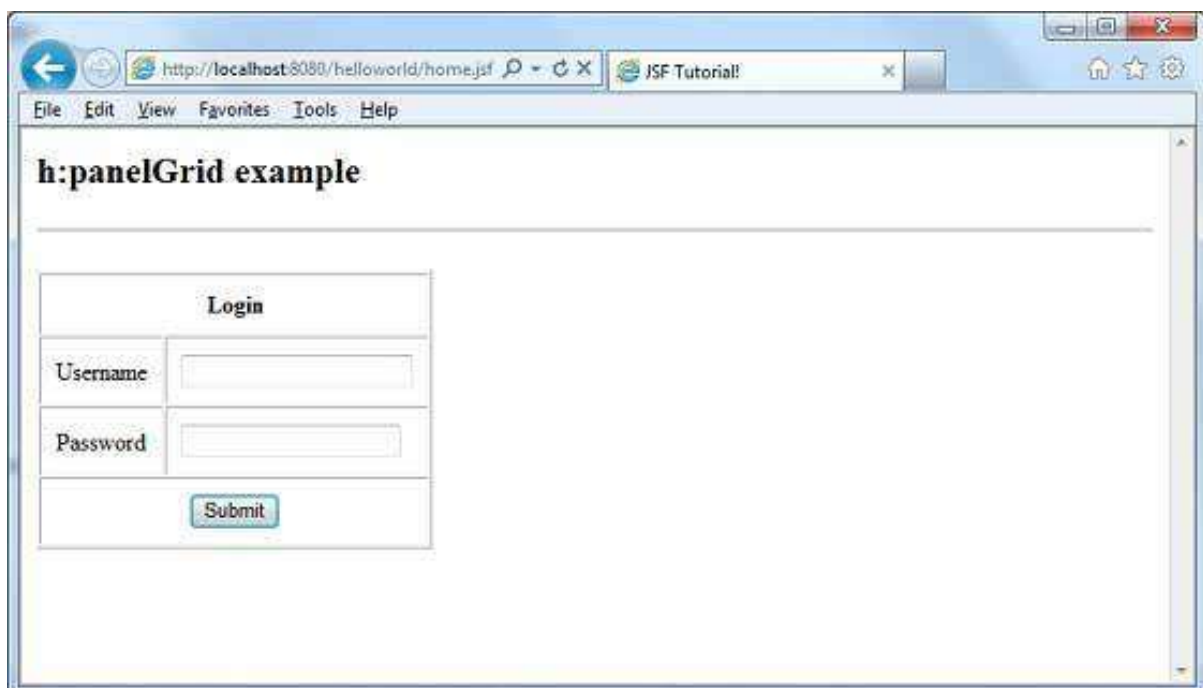
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:panelGrid example</h2>
    <hr />
    <h:form>
```

```

<h:panelGrid id="panel" columns="2" border="1"
    cellpadding="10" cellspacing="1">
    <f:facet name="header">
        <h:outputText value="Login"/>
    </f:facet>
    <h:outputLabel value="Username" />
    <h:inputText />
    <h:outputLabel value="Password" />
    <h:inputSecret />
    <f:facet name="footer">
        <h:panelGroup style="display:block; text-align:center">
            <h:commandButton id="submit" value="Submit" />
        </h:panelGroup>
    </f:facet>
</h:panelGrid>
</h:form>
</body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.





## h:message

The h:message tag displays message corresponding to UI element.

### JSF Tag

```
<h:inputText id="username" size="20" label="UserName" required="true">
    <f:validateLength for="username" minimum="5" maximum="20" />
</h:inputText>
<h:message for="username" style="color:red" />
```

### Rendered Output

In case the username entered is more than 20 characters.

```
<span style="color:red">UserName: Validation Error:
Length is greater than allowable maximum of '20'</span>
```

In case the username entered is less than 5 characters.

```
<span style="color:red">UserName: Validation Error:
Length is less than allowable minimum of '5'</span>
```

In case the username is not entered.

```
<span style="color:red">UserName: Validation Error:
Value is required</span>
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b>

	Cascading stylesheet (CSS) class name
5	<b>for</b>  The ID of the component whose message is displayed, applicable only to h:message
6	<b>errorClass</b> CSS class applied to error messages
7	<b>errorStyle</b> CSS style applied to error messages
8	<b>fatalClass</b> CSS class applied to fatal messages
9	<b>fatalStyle</b> CSS style applied to fatal messages
10	<b>globalOnly</b>  Instruction to display only global messages, applicable only to h:messages. Default: false
11	<b>infoClass</b> CSS class applied to information messages
12	<b>infoStyle</b> CSS style applied to information messages
13	<b>layout</b> Specification for message layout: table or list, applicable only to h:messages
14	<b>showDetail</b>  A boolean that determines whether message details are shown. Defaults are false for h:messages, true for h:message
15	<b>showSummary</b>

	A boolean that determines whether message summaries are shown. Defaults are true for h:messages, false for h:message
16	<b>tooltip</b>  A boolean that determines whether message details are rendered in a tooltip; the tooltip is only rendered if showDetail and showSummary are true
17	<b>warnClass</b> CSS class for warning messages
18	<b>warnStyle</b> CSS style for warning messages
19	<b>style</b> Inline style information
20	<b>title</b>  A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```

<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:messages example</h2>
    <hr />
    <h:form>
        <h:panelGrid id="panel" columns="3" border="0" cellpadding="10"
            cellspacing="1">
            <h:outputLabel value="Enter Username" />
            <h:inputText id="username" size="20" label="UserName"
                required="true">
                <f:validateLength for="username" minimum="5" maximum="20" />
            </h:inputText>
            <h:message for="username" style="color:red" />
            <h:outputLabel value="Enter Password" />
            <h:inputSecret id="password" size="20" label="Password"
                required="true" reDisplay="true" >
                <f:validateLength for="password" minimum="5" maximum="10" />
            </h:inputSecret>
            <h:message for="password" style="color:red" />
            <h:commandButton id="submit" value="Submit" action="result"/>
        </h:panelGrid>
    </h:form>
</body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## h:messages

The h:messages tag shows all the messages at one place corresponding to UI elements.

### JSF Tag

```
<h:messages style="color:red;margin:8px;" />
```

### Rendered Output

Case: Username entered is more than 20 characters and password entered is less than 5 characters.

```
<ul style="color:red;margin:8px;">
  <li>    UserName: Validation Error:
    Length is greater than allowable maximum of '20' </li>
  <li>    Password: Validation Error:
    Length is less than allowable minimum of '5' </li>
</ul>
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>rendered</b> A boolean; false suppresses rendering
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>for</b> The ID of the component whose message is displayed, applicable only to h:message
6	<b>errorClass</b> CSS class applied to error messages
7	<b>errorStyle</b> CSS style applied to error messages
8	<b>fatalClass</b> CSS class applied to fatal messages
9	<b>fatalStyle</b> CSS style applied to fatal messages

10	<b>globalOnly</b> Instruction to display only global messages, applicable only to h:messages. Default: false
11	<b>infoClass</b> CSS class applied to information messages
12	<b>infoStyle</b> CSS style applied to information messages
13	<b>layout</b> Specification for message layout: table or list, applicable only to h:messages
14	<b>showDetail</b> A boolean that determines whether message details are shown. Defaults are false for h:messages, true for h:message
15	<b>showSummary</b> A boolean that determines whether message summaries are shown. Defaults are true for h:messages, false for h:message
16	<b>tooltip</b> A boolean that determines whether message details are rendered in a tooltip; the tooltip is only rendered if showDetail and showSummary are true
17	<b>warnClass</b> CSS class for warning messages
18	<b>warnStyle</b> CSS style for warning messages
19	<b>style</b> Inline style information

20	<p><b>title</b></p> <p>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value</p>
----	--

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
```



```

<body>
  <h2>h:messages example</h2>
  <hr />
  <h:form>
    <h:panelGrid id="panel" columns="2" border="0" cellpadding="10"
      cellspacing="1">
      <h:outputLabel value="Enter Username" />
      <h:inputText id="username" size="20" label="UserName"
        required="true">
        <f:validateLength for="username" minimum="5" maximum="20" />
      </h:inputText>
      <h:outputLabel value="Enter Password" />
      <h:inputSecret id="password" size="20" label="Password"
        required="true" reDisplay="true" >
        <f:validateLength for="password" minimum="5" maximum="10" />
      </h:inputSecret>
      <h:commandButton id="submit" value="Submit" action="result"/>
    </h:panelGrid>
    <h:messages style="color:red;margin:8px;" />
  </h:form>
</body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## f:param

f:param tag provides the options to pass parameters to a component or pass request parameters.

### JSF Tag

Pass parameter to a UI component

```
<h:outputFormat value="Hello {0}!.">
    <f:param value="World" />
</h:outputFormat>
```

Pass request parameter

```
<h:commandButton id="submit"
    value="Show Message" action="#{userData.showResult}">
    <f:param name="username" value="JSF 2.0 User" />
</h:commandButton>
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>binding</b> Reference to the component that can be used in a backing bean
3	<b>name</b> An optional name for this parameter component
4	<b>value</b> The value stored in this component

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### UserData.java

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

```

import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

    public String showResult(){
        FacesContext fc = FacesContext.getCurrentInstance();
        Map<String,String> params =
        fc.getExternalContext().getRequestParameterMap();
        data = params.get("username");
        return "result";
    }
}

```

### home.xhtml

```

<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>f:param example</h2>
    <hr />

```

```

<h:form>
  <h:outputFormat value="Hello {0}!.">
    <f:param value="World" />
  </h:outputFormat>
  <br/>
  <h:commandButton id="submit"
    value="Show Message" action="#{userData.showResult}">
    <f:param name="username" value="JSF 2.0 User" />
  </h:commandButton>
</h:form>
</body>
</html>

```

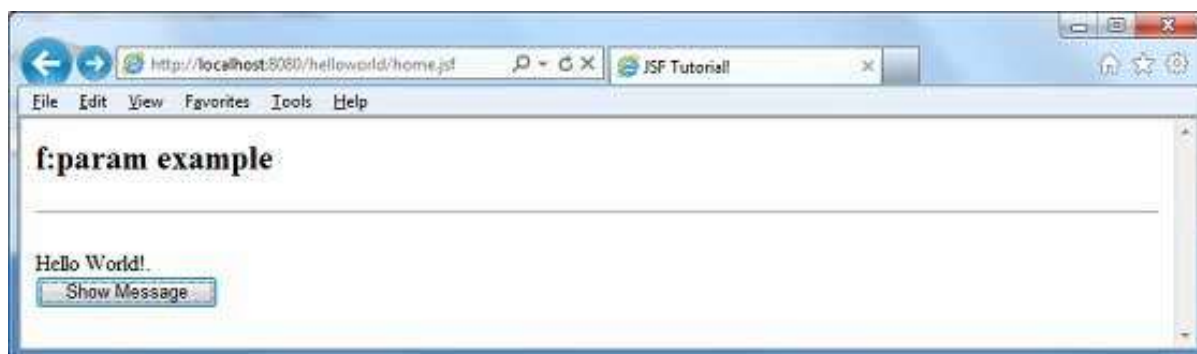
### result.xhtml

```

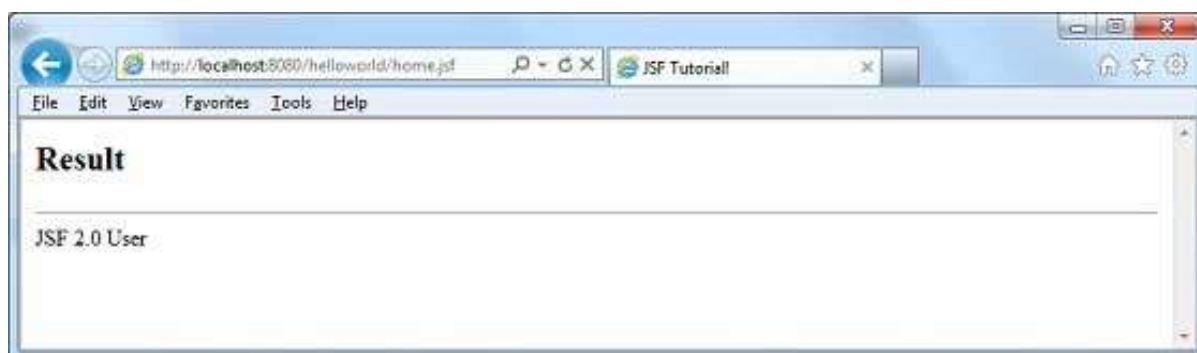
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <head>
    <title>JSF Tutorial!</title>
  </head>
  <h:body>
    <h2>Result</h2>
    <hr />
    <div>
      <h:outputText value="#{userData.data}" />
    </div>
  </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Press **Show Message** button and you'll see the following result.



## f:attribute

The h:attribute tag provides option to pass a attribute value to a component, or a parameter to a component via action listener.

## JSF Tag

```
<h:commandButton id="submit"
  actionListener="#{userData.attributeListener}" action="result">
  <f:attribute name="value" value="Show Message" />
  <f:attribute name="username" value="JSF 2.0 User" />
</h:commandButton>
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>name</b> The name of the attribute to set
2	<b>value</b> The value of the attribute

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### UserData.java

```
package com.tutorialspoint.test;

import java.io.Serializable;
```

```

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

    public void attributeListener(ActionEvent event){
        data = (String)event.getComponent().getAttributes().get("username");
    }
}

```

### home.xhtml

```

<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>f:attribute example</h2>
    <hr />
    <h:form>
        <h:commandButton id="submit"
            actionListener="#{userData.attributeListener}" action="result">
            <f:attribute name="value" value="Show Message" />

```



```

        <f:attribute name="username" value="JSF 2.0 User" />
    </h:commandButton>
</h:form>
</body>
</html>

```

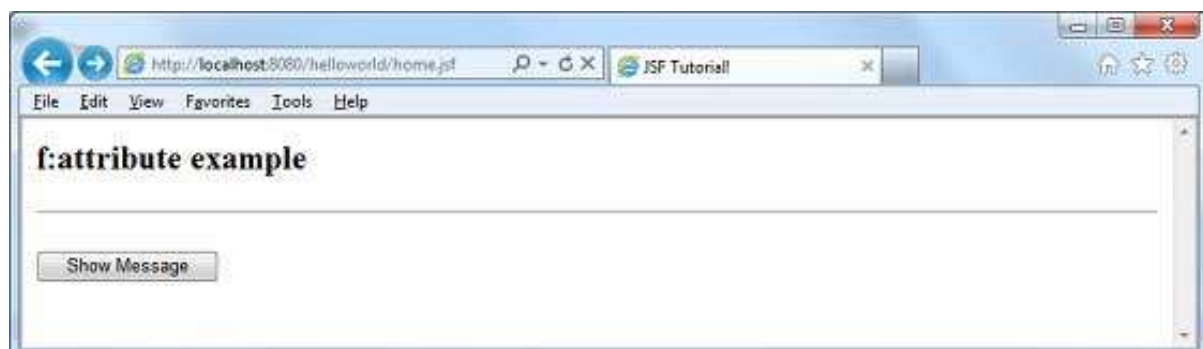
### result.xhtml

```

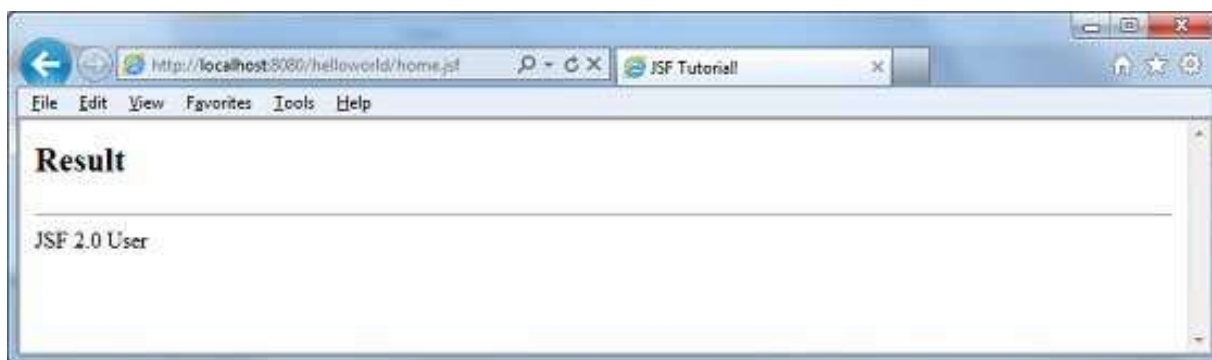
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
        <h2>Result</h2>
        <hr />
        #{userData.data}
    </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Press **Show Message** button and you'll see the following result.



## h:setPropertyActionListener

The `h:setPropertyActionListener` tag adds an action listener to a component that sets a bean property to a given value.

### JSF Tag

```
<h:commandButton id="submit" action="result" value="Show Message">
    <f:setPropertyActionListener target="#{userData.data}"
        value="JSF 2.0 User" />
</h:commandButton>
```

### Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.

5	Compile and run the application to make sure business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### **UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

}
```

**home.xhtml**

```

<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>f:attribute example</h2>
    <hr />
    <h:form>
        <h:commandButton id="submit" action="result" value="Show Message">
            <f:setPropertyActionListener
                target="#{userData.data}" value="JSF 2.0 User" />
        </h:commandButton>
    </h:form>
</body>
</html>

```

**result.xhtml**

```

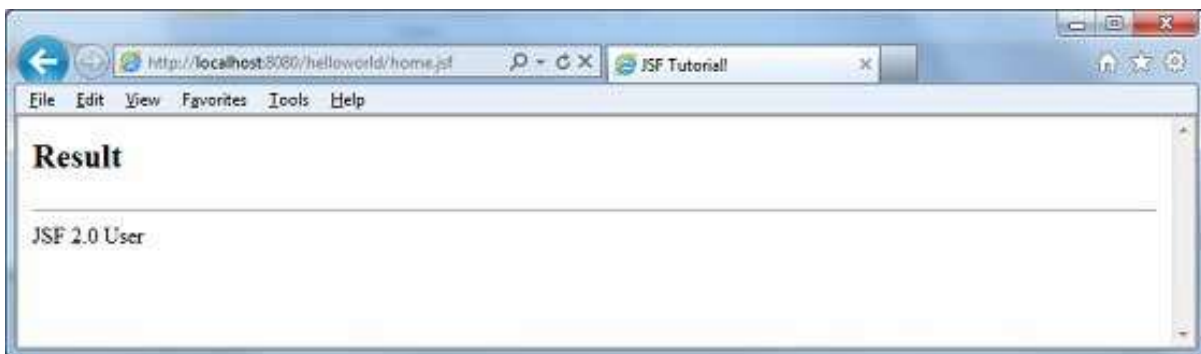
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
    <title>JSF Tutorial!</title>
</head>
<h:body>
    <h2>Result</h2>
    <hr />
    <#{userData.data}>
</h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Press **Show Message** button and you'll see the following result.



## 9. JSF – Facelet Tags

JSF provides special tags to create common layout for a web application called facelets tags. These tags provide flexibility to manage common parts of 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 2.0.

Sr. No.	Tag & Description
1	<b><u>Templates</u></b>  We'll demonstrate how to use templates using the following tags <ul style="list-style-type: none"><li>• &lt;ui:insert&gt;</li><li>• &lt;ui:define&gt;</li><li>• &lt;ui:include&gt;</li><li>• &lt;ui:composition&gt;</li></ul>
2	<b><u>Parameters</u></b>  We'll demonstrate how to pass parameters to a template file using the following tag <ul style="list-style-type: none"><li>• &lt;ui:param&gt;</li></ul>

3	<b><u>Custom</u></b> We'll demonstrate how to create custom tags
4	<b><u>Remove</u></b> We'll demonstrate capability to remove JSF code from generated HTML page

## Template Tags

Templates in a web application defines a common interface layout and style. For example, a same banner, logo in common header and copyright information in footer. JSF provides following facet tags to provide a standard web interface layout.

Sr. No.	Tag & Description
1	<b>ui:insert</b> Used in template file. It defines contents to be placed in a template. ui:define tag can replaced its contents.
2	<b>ui:define</b> Defines the contents to be inserted in a template.
3	<b>ui:include</b> Includes contents of one xhtml page into another xhtml page.
4	<b>ui:composition</b> Loads a template using <b>template</b> attribute. It can also define a group of components to be inserted in xhtml page.

## Creating Template

---

Creating template for a web application is a step-by-step procedure. Following are the steps to create a sample template.

### Step 1: Create Header file: header.xhtml

Use **ui:composition** tag to define a default content of Header section.

```
<ui:composition>
    <h1>Default Header</h1>
</ui:composition>
```

### Step 2: Create Footer file: footer.xhtml

Use **ui:composition** tag to define a default content of Footer section.

```
<ui:composition>
    <h1>Default Footer</h1>
</ui:composition>
```

### Step 3: Create Content file: contents.xhtml

Use **ui:composition** tag to define a default content of Content section.

```
<ui:composition>
    <h1>Default Contents</h1>
</ui:composition>
```

### Step 4: Create a Template: common.xhtml

Use **ui:insert** and **ui:include** tag to include header/footer and content file in template file. Name each section in **ui:insert** tag.

**name** attribute of **ui:insert** tag will be used to replace the contents of the corresponding section.

```
<h:body>
    <ui:insert name="header" >
        <ui:include src="header.xhtml" />
    </ui:insert>
    <ui:insert name="content" >
        <ui:include src="contents.xhtml" />
    </ui:insert>
    <ui:insert name="footer" >
        <ui:include src="footer.xhtml" />
    </ui:insert>
```



```

    </ui:insert>
</h:body>

```

### Step 5a: Use Template with default contents: home.xhtml

Load common.xhtml, a template using **ui:composition** tag in any xhtml page.

```

<h:body>
    <ui:composition template="common.xhtml">
</h:body>

```

### Step 5b: Use Template and set own contents: home.xhtml

Load common.xhtml, a template using **ui:composition** tag in any xhtml page. Use **ui:define** tag to override default values.

```

<h:body>
    <ui:composition template="templates/common.xhtml">
        <ui:define name="content">
            <h:link value="Page 1" outcome="page1" />
            &nbsp;
            <h:link value="Page 2" outcome="page2" />
        </ui:define>
    </ui:composition>
</h:body>

```

## Example Application

Let us create a test JSF application to test the template tags in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.

2	Create <i>templates</i> folder under <i>src &gt; main &gt; webapp</i> folder.
3	Create <i>header.xhtml</i> , <i>footer.xhtml</i> , <i>contents.xhtml</i> and <i>common.xhtml</i> files under <i>src &gt; main &gt; webapp &gt; templates</i> folder. Modify them as explained below.
4	Create <i>page1.xhtml</i> and <i>page2.xhtml</i> files under <i>src &gt; main &gt; webapp</i> folder. Modify them as explained below.
5	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
6	Compile and run the application to make sure business logic is working as per the requirements.
7	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
8	Launch your web application using appropriate URL as explained below in the last step.

### header.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets">
  <body>
    <ui:composition>
      <h1>Default Header</h1>
    </ui:composition>
  </body>
```

```
</html>
```

### footer.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <body>
    <ui:composition>
      <h1>Default Footer</h1>
    </ui:composition>
  </body>
</html>
```

### contents.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <body>
    <ui:composition>
      <h1>Default Content</h1>
    </ui:composition>
  </body>
</html>
```

### common.xhtml

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

```

xmlns:ui="http://java.sun.com/jsf/facelets">
<h:head></h:head>
<h:body>
  <div style="border-width:2px; border-color:green; border-style:solid;">
    <ui:insert name="header" >
      <ui:include src="/templates/header.xhtml" />
    </ui:insert>
  </div>
  <br/>
  <div style="border-width:2px; border-color:black; border-style:solid;">
    <ui:insert name="content" >
      <ui:include src="/templates/contents.xhtml" />
    </ui:insert>
  </div>
  <br/>
  <div style="border-width:2px; border-color:red; border-style:solid;">
    <ui:insert name="footer" >
      <ui:include src="/templates/footer.xhtml" />
    </ui:insert>
  </div>
</h:body>
</html>

```

### page1.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:body>
    <ui:composition template="templates/common.xhtml">
      <ui:define name="header">
        <h2>Page1 header</h2>
      </ui:define>
      <ui:define name="content">

```

```

        <h2>Page1 content</h2>
        <h:link value="Back To Home" outcome="home" />
    </ui:define>
    <ui:define name="footer">
        <h2>Page1 Footer</h2>
    </ui:define>
</ui:composition>
</h:body>
</html>

```

### page2.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
    <h:body>
        <ui:composition template="templates/common.xhtml">
            <ui:define name="header">
                <h2>Page2 header</h2>
            </ui:define>
            <ui:define name="content">
                <h2>Page2 content</h2>
                <h:link value="Back To Home" outcome="home" />
            </ui:define>
            <ui:define name="footer">
                <h2>Page2 Footer</h2>
            </ui:define>
        </ui:composition>
    </h:body>
</html>

```

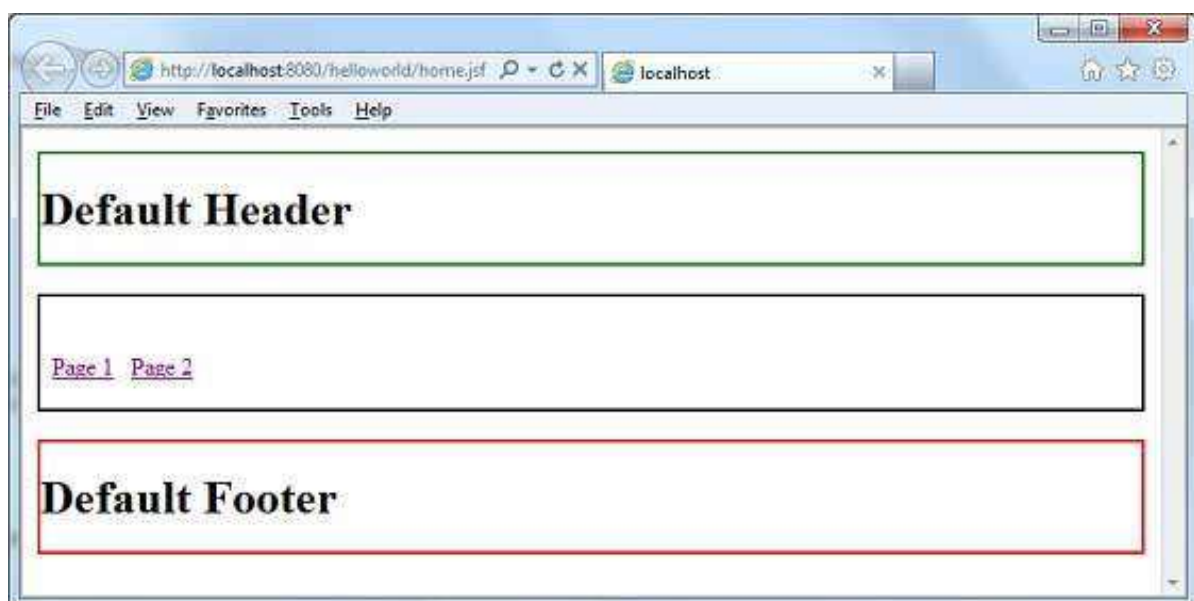
**home.xhtml**

```

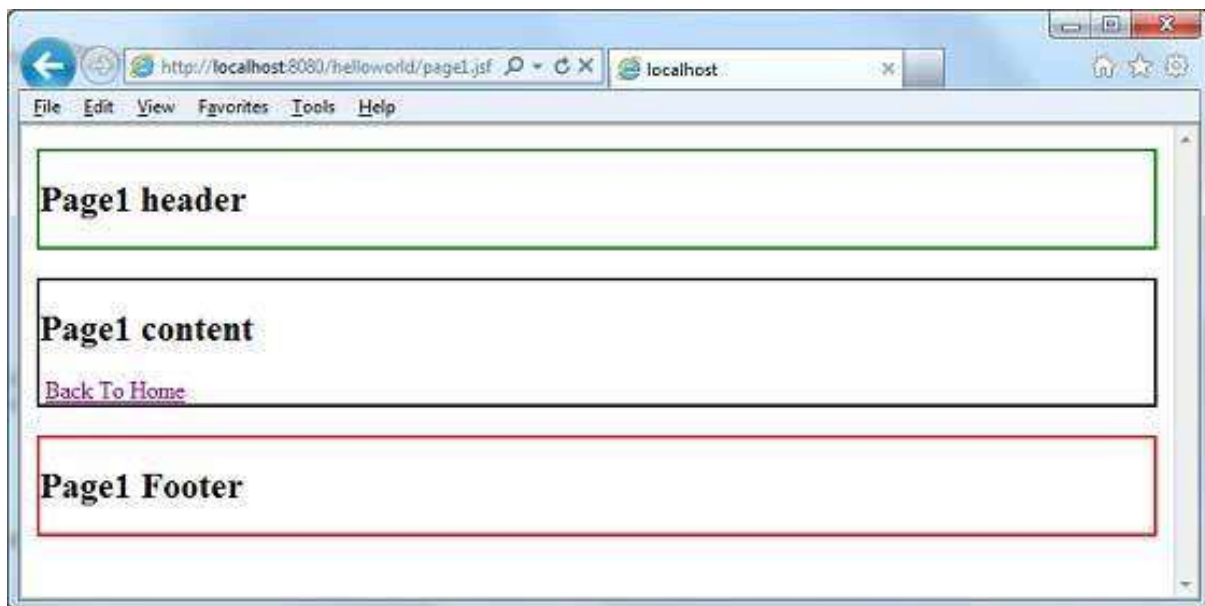
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:body>
    <ui:composition template="templates/common.xhtml">
      <ui:define name="content">
        <br/><br/>
        <h:link value="Page 1" outcome="page1" />
        <h:link value="Page 2" outcome="page2" />
        <br/><br/>
      </ui:define>
    </ui:composition>
  </h:body>
</html>

```

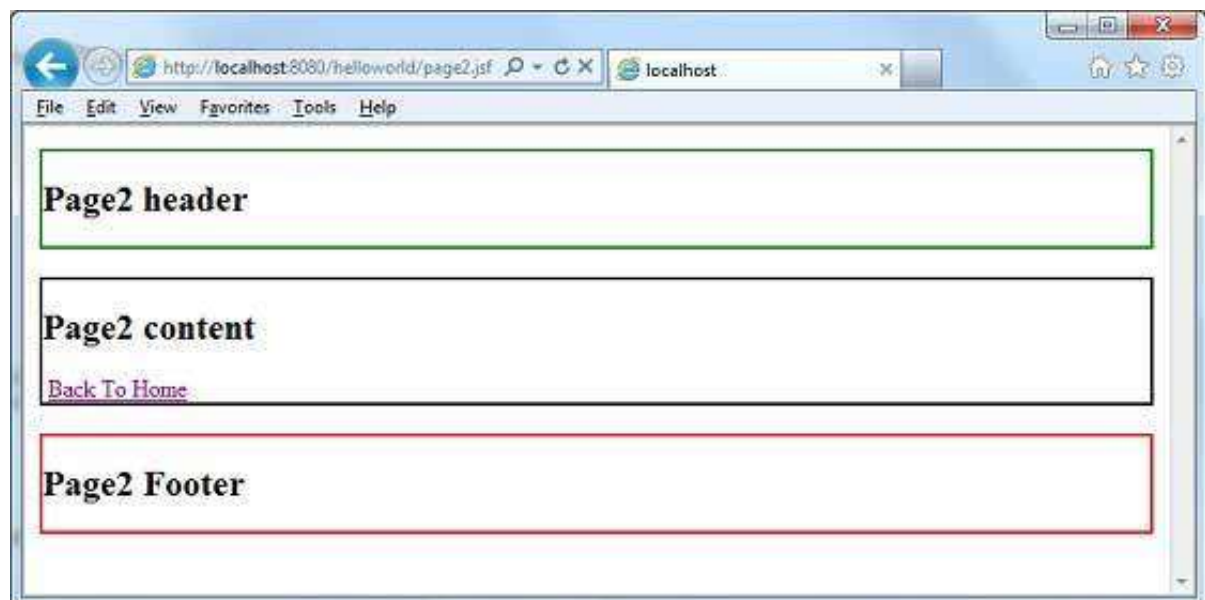
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Click **Page1** link and you'll see the following result.



Or Click **Page2** link and you'll see the following result.



## ui:param Tag

Using ui:param tag, we can pass parameters to template file or an included file.

In *JSF - template tags* chapter, we've learned how to create and use template tags. We defined various section such as header, footer, content, and a template combining all the sections.

Now we'll learn -

- How to pass parameter(s) to various section of a template
- How to pass parameter(s) to a template

## Parameter to Section of a Template

---

### Create parameter : common.xhtml

Add parameter to ui:include tag. Use **ui:param** tag to define a parameter containing a value to be passed to Header section.

```
<ui:insert name="header" >
    <ui:include src="/templates/header.xhtml" >
        <ui:param name="defaultHeader" value="Default Header" />
    </ui:include>
</ui:insert>
```

### Use parameter : header.xhtml

```
<ui:composition>
    <h1>#{defaultHeader}</h1>
</ui:composition>
```

## Parameter to Template

---

### Create parameter : home.xhtml

Add parameter to ui:composition tag. Use **ui:param** tag to define a parameter containing a value to be passed to template.

```
<ui:composition template="templates/common.xhtml">
    <ui:param name="title" value="Home" />
</ui:composition>
```

### Use parameter : common.xhtml

```
<h:body>
    <h2>#{title}</h2>
</h:body>
```



## Example Application

Let us create a test JSF application to test the template tags in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - Templates Tag</i> chapter.
2	Modify <i>header.xhtml</i> , and <i>common.xhtml</i> files under <i>src &gt; main &gt; webapp &gt; templates</i> folder. Modify them as explained below.
3	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
4	Compile and run the application to make sure business logic is working as per the requirements.
5	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
6	Launch your web application using appropriate URL as explained below in the last step.

### header.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <body>
    <ui:composition>
      <h1>#{defaultHeader}</h1>
```

```

    </ui:composition>
  </body>
</html>

```

### common.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head></h:head>
  <h2>#{title}</h2>
  <div style="border-width:2px; border-color:green; border-style:solid;">
    <ui:insert name="header" >
      <ui:include src="/templates/header.xhtml" >
        <ui:param name="defaultHeader" value="Default Header" />
      </ui:include>
    </ui:insert>
  </div>
  <br/>
  <div style="border-width:2px; border-color:black; border-style:solid;">
    <ui:insert name="content" >
      <ui:include src="/templates/contents.xhtml" />
    </ui:insert>
  </div>
  <br/>
  <div style="border-width:2px; border-color:red; border-style:solid;">
    <ui:insert name="footer" >
      <ui:include src="/templates/footer.xhtml" />
    </ui:insert>
  </div>
</h:body>
</html>

```

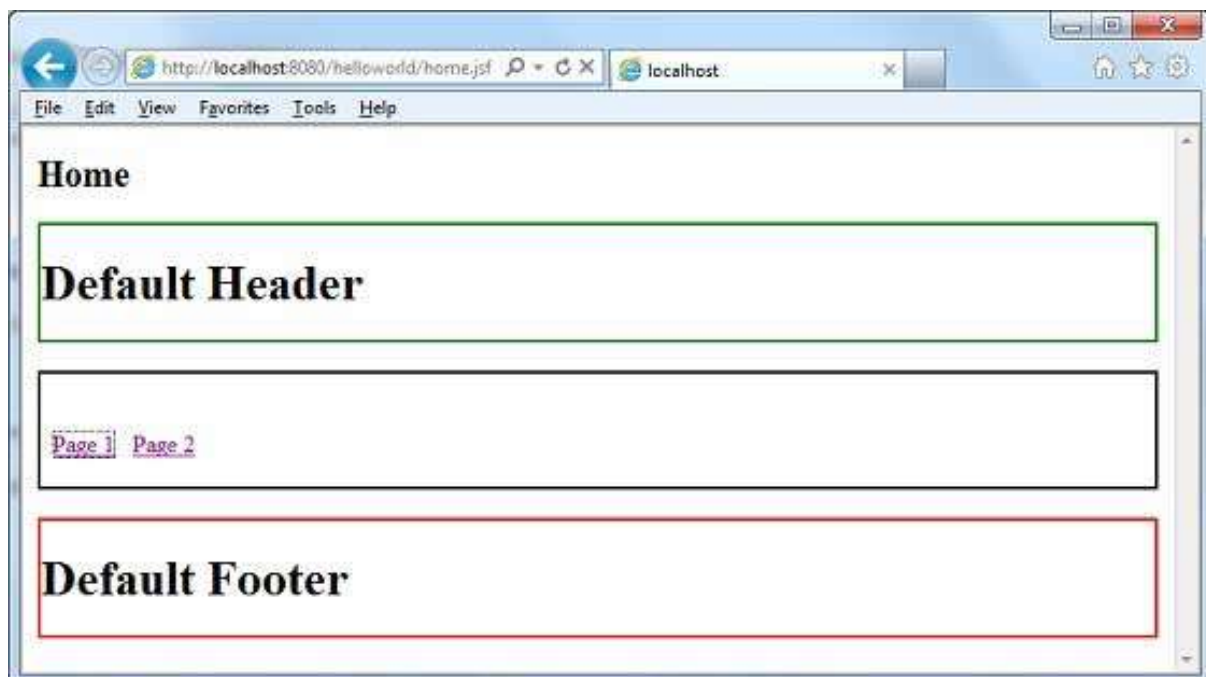
### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:body>
    <ui:composition template="templates/common.xhtml">
      <ui:param name="title" value="Home" />
      <ui:define name="content">
        <br/><br/>
        <h:link value="Page 1" outcome="page1" />
        <h:link value="Page 2" outcome="page2" />
        <br/><br/>
      </ui:define>
    </ui:composition>
  </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## Custom Tag

JSF provides the developer with a powerful capability to define own custom tags, which can be used to render custom contents.

Defining a custom tag in JSF is a three-step process.

Step No.	Description
1a	Create a xhtml file and define contents in it using <b>ui:composition</b> tag
1b	Create a tag library descriptor (.taglib.xml file) and declares the above custom tag in it.
1c	Register the tag library descriptor in web.xml

### Step 1a: Define custom tag contents : buttonPanel.xhtml

```
<h:body>
    <ui:composition>
        <h:commandButton type="submit" value="#{okLabel}" />
        <h:commandButton type="reset" value="#{cancelLabel}" />
    </ui:composition>
</h:body>
```

### Step 1b: Define a tag library : tutorialspoint.taglib.xml

As the name mentions a Tag library is a library of tags. Following table describes important attributes of a tag library.

Sr. No.	Node & Description
1	<b>facelet-taglib</b> Contains all the tags.
2	<b>namespace</b> Namespace of the tag library and should be unique.

3	<b>tag</b> Contains a single tag
4	<b>tag-name</b> Name of the tag
5	<b>source</b> Tag implementation

```
<facelet-taglib>
  <namespace>http://tutorialspoint.com/facelets</namespace>
  <tag>
    <tag-name>buttonPanel</tag-name>
    <source>com/tutorialspoint/buttonPanel.xhtml</source>
  </tag>
</facelet-taglib>
```

### Step 1c: Register the tag library :web.xml

```
<context-param>
  <param-name>javax.faces.FACELETS_LIBRARIES</param-name>
  <param-value>/WEB-INF/tutorialspoint.taglib.xml</param-value>
</context-param>
```

Using a custom tag in JSF is a two-step process.

Step No.	Description
2a	Create a xhtml file and use custom tag library's namespace
2b	Use the custom tag as normal JSF tags

## Step 2a: Use Custom Namespace: home.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  xmlns:tp="http://tutorialspoint.com/facelets">
```

## Step 2b: Use Custom Tag: home.xhtml

```
<h:body>
  <tp:buttonPanel okLabel="Ok" cancelLabel="Cancel" />
</h:body>
```

## Example Application

Let us create a test JSF application to test the template tags in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Create <i>com</i> folder under <i>WEB-INF</i> directory.
3	Create <i>tutorialspoint</i> folder under <i>WEB-INF &gt; com</i> directory.
4	Create <i>buttonPanel.xhtml</i> file under <i>WEB-INF &gt; com &gt; tutorialspoint</i> folder. Modify it as explained below.
5	Create <i>tutorialspoint.taglib.xml</i> file under <i>WEB-INF</i> folder. Modify it as explained below.
6	Modify <i>web.xml</i> file under <i>WEB-INF</i> folder as explained below.

7	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
8	Compile and run the application to make sure business logic is working as per the requirements.
9	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
10	Launch your web application using appropriate URL as explained below in the last step.

### buttonPanel.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:body>
    <ui:composition>
      <h:commandButton type="submit" value="#{okLabel}" />
      <h:commandButton type="reset" value="#{cancelLabel}" />
    </ui:composition>
  </h:body>
</html>
```

### tutorialspoint.taglib.xml

```

<?xml version="1.0"?>
<!DOCTYPE facelet-taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//EN"
"http://java.sun.com/dtd/facelet-taglib_1_0.dtd">
<facelet-taglib>
  <namespace>http://tutorialspoint.com/facelets</namespace>
  <tag>
    <tag-name>buttonPanel</tag-name>
    <source>com/tutorialspoint/buttonPanel.xhtml</source>
  </tag>
</facelet-taglib>

```

### web.xml

```

<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <context-param>
    <param-name>javax.faces.FACELETS_LIBRARIES</param-name>
    <param-value>/WEB-INF/tutorialspoint.taglib.xml</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
</web-app>

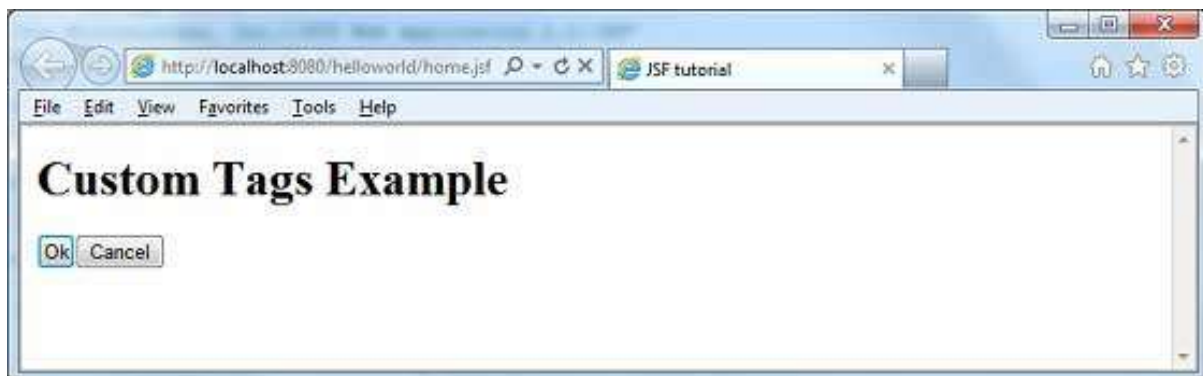
```

### home.xhtml



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:tp="http://tutorialspoint.com/facelets">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h1>Custom Tags Example</h1>
    <tp:buttonPanel okLabel="Ok" cancelLabel="Cancel" />
  </h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## ui:remove Tag

ui:remove tag is used to prevent the JSF specific code to be rendered on the client side. It is used especially to prevent commented out code to be rendered on the client side.

## JSF Tag Commented Out Using HTML Comment

```
<!-- JSF code commented out -->
<!--
<h:commandButton value="Ok" />
-->
```

## Rendered Output

```
<!-- JSF code commented out -->
<!--
<h:commandButton value="Ok" />
-->
```

Now using remove tag we'll see the following change in rendered output.

## JSF Tag Commented Out Using Remove Tag

```
<!-- JSF code commented out -->
<ui:remove>
    <h:commandButton value="Ok" />
</ui:remove>
```

## Rendered Output

```
<!-- JSF code commented out -->
```

## Example Application

Let us create a test JSF application to test the template tags in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

5	Launch your web application using appropriate URL as explained below in the last step.
---	--

### home.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <ui:remove>
      <h:commandButton value="Ok" />
    </ui:remove>
    <!--
      <h:commandButton value="Cancel" />
    -->
  </h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, you'll see an empty page.

View source of the page and you will see the following html text.

### home.jsf

```
<?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"><head>
  <title>JSF tutorial</title>
</head>
<body>
```

```
<!--  
    <h:commandButton value="Cancel" />  
-->  
</body>  
</html>
```

# 10. JSF - Converter Tags

JSF provides inbuilt converters 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 *Converter Tags* in JSF 2.0:

Sr. No.	Tag & Description
1	<b><u>f:convertNumber</u></b> Converts a String into a Number of desired format
2	<b><u>f:convertDateTime</u></b> Converts a String into a Date of desired format
3	<b><u>Custom Converter</u></b> Creating a custom convertor

## **f:convertNumber**

f:convertNumber tag is used to convert a string value to a number of required format.

### **JSF Tag**

```
<f:convertNumber minFractionDigits="2" />
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>type</b> number (default), currency, or percent
2	<b>pattern</b> Formatting pattern, as defined in java.text.DecimalFormat
3	<b>maxFractionDigits</b> Maximum number of digits in the fractional part
4	<b>minFractionDigits</b> Minimum number of digits in the fractional part
5	<b>maxIntegerDigits</b> Maximum number of digits in the integer part
6	<b>minIntegerDigits</b> Minimum number of digits in the integer part
7	<b>integerOnly</b> True, if only the integer part is parsed (default: false)
8	<b>groupingUsed</b> True, if grouping separators are used (default: true)
9	<b>locale</b> Locale whose preferences are to be used for parsing and formatting
10	<b>currencyCode</b> ISO 4217 currency code to use when converting currency values
11	<b>currencySymbol</b> Currency symbol to use when converting currency values

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

### home.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>ConvertNumber Example</h2>
```

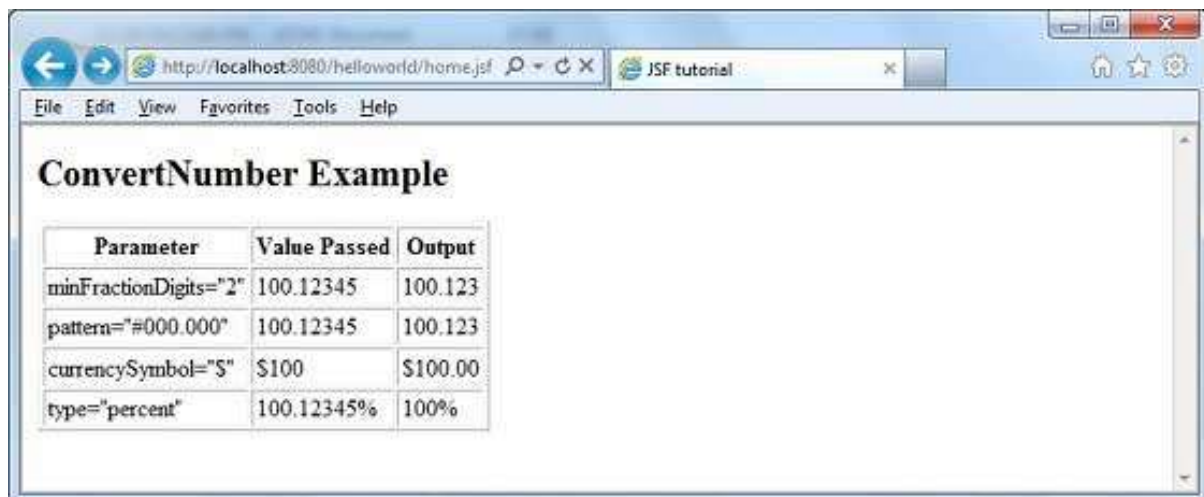
```

<table border="1" cellspacing="2" cellpadding="2">
  <tr><th>Parameter</th><th>Value Passed</th><th>Output</th></tr>
  <tr><td>minFractionDigits="2"</td><td>100.12345</td>
  <td>
    <h:outputText value="100.12345" >
      <f:convertNumber minFractionDigits="2" />
    </h:outputText>
  </td></tr>
  <tr><td>pattern="#000.000"</td><td>100.12345</td>
  <td>
    <h:outputText value="100.12345" >
      <f:convertNumber pattern="#000.000" />
    </h:outputText>
  </td></tr>
  <tr><td>currencySymbol="$"</td><td>$100</td>
  <td>
    <h:outputText value="$100">
      <f:convertNumber currencySymbol="$" type="currency" />
    </h:outputText>
  </td></tr>
  <tr><td>type="percent"</td><td>100.12345%</td>
  <td>
    <h:outputText value="100.12345%" >
      <f:convertNumber type="percent" />
    </h:outputText>
  </td></tr>
</table>
</h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.





## f:convertDateTime

f:convertDateTime tag is used to convert a string value to a date of required format. It also acts as a validator, a required date format.

### JSF Tag

```
<f:convertDateTime pattern="dd-mm-yyyy" />
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>type</b> date (default), time, or both
2	<b>dateStyle</b> default, short, medium, long, or full
3	<b>timeStyle</b> default, short, medium, long, or full

4	<b>pattern</b> Formatting pattern, as defined in <code>java.text.SimpleDateFormat</code>
5	<b>locale</b> Locale whose preferences are to be used for parsing and formatting
6	<b>timeZone</b> Time zone to use for parsing and formatting

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure business logic is working as per the requirements.

6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### **UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.Date;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public Date date;

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>ConvertDateTime Example</h2>
    <h:form>
      <h:inputText id="dateInput" value="#{userData.date}"
        label="Date" >
        <f:convertDateTime pattern="dd-mm-yyyy" />
      </h:inputText>
      <h:commandButton value="submit" action="result"/>
    </h:form>
    <br/>
    <h:message for="dateInput" style="color:red" />
    <h:outputText value="12-01-2012" >
      <f:convertDateTime pattern="dd-mm-yyyy" />
    </h:outputText>
  </h:body>
</html>

```

**result.xhtml**

```

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

```

```

    <h2>Result</h2>
    <hr />
    #{userData.date}
</h:body>
</html>

```

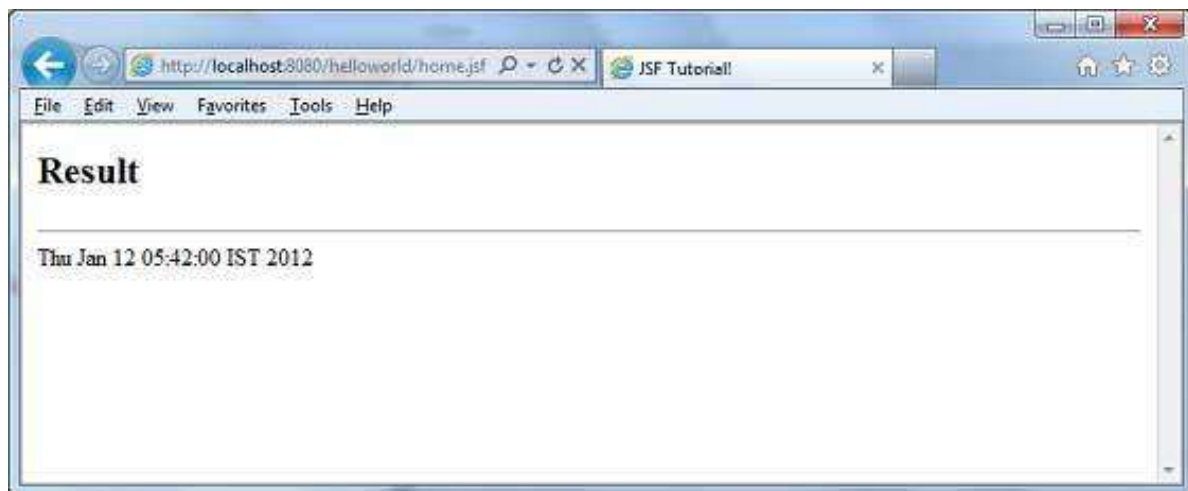
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Enter any invalid value and press Submit button. See the following error message.



Enter any valid value and press Submit button. See the following result.



## Custom Converter

We can create our own Custom convertor in JSF.

Defining a custom converter in JSF is a three-step process.

Step No.	Description
1	Create a converter class by implementing <i>javax.faces.convert.Converter</i> interface.
2	Implement <i>getAsObject()</i> and <i>getAsString()</i> methods of above interface.
3	Use Annotation <i>@FacesConverter</i> to assign a unique id to the custom convertor.

### Step 1: Create a Converter Class : *UrlConverter.java*

```
public class UrlConverter implements Converter {
    ...
}
```

## Step 2: Implement Converter Interface Methods : UrlConverter.java

Create a simple class to store data: UrlData. This class will store a URL string.

```
public class UrlData {  
  
    private String url;  
  
    public UrlData(String url){  
        this.url = url;  
    }  
    ...  
}
```

Use UrlData in getAsObject method.

```
public class UrlConverter implements Converter {  
    @Override  
    public Object getAsObject(FacesContext facesContext,  
        UIComponent component, String value) {  
        ...  
        UrlData urlData = new UrlData(url.toString());  
        return urlData;  
    }  
  
    @Override  
    public String getAsString(FacesContext facesContext,  
        UIComponent component, Object value) {  
        return value.toString();  
    }  
}
```

## Step 3: Annotate to Register the Convertor : UrlConverter.java

```
@FacesConverter("com.tutorialspoint.test.UrlConverter")  
public class UrlConverter implements Converter {  
}
```

## Use the Converter in JSF Page

```
<h:inputText id="urlInput" value="#{userData.data}" label="URL" >
    <f:converter converterId="com.tutorialspoint.test.UrlConverter" />
</h:inputText>
```

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Create <i>UrlData.java</i> under package <i>com.tutorialspoint.test</i> as explained below.
3	Create <i>UrlConvertor.java</i> as a converter under package <i>com.tutorialspoint.test</i> as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Modify <i>home.xhtml</i> as explained below. Keep rest of the files unchanged.
6	Create <i>result.xhtml</i> in the webapps directory as explained below.



7	Compile and run the application to make sure the business logic is working as per the requirements.
8	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
9	Launch your web application using appropriate URL as explained below in the last step.

### UrlData.java

```
package com.tutorialspoint.test;

public class UrlData {
    private String url;

    public UrlData(String url){
        this.url = url;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String toString(){
        return url;
    }
}
```

**UrlConverter.java**

```
package com.tutorialspoint.test;

import java.net.URI;
import java.net.URISyntaxException;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;
import javax.faces.convert.FacesConverter;

@FacesConverter("com.tutorialspoint.test.UrlConverter")
public class UrlConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext facesContext,
        UIComponent component, String value) {

        StringBuilder url = new StringBuilder();

        if(!value.startsWith("http://", 0)){
            url.append("http://");
        }
        url.append(value);

        try {
            new URI(url.toString());
        } catch (URISyntaxException e) {
            FacesMessage msg = new FacesMessage("Error converting URL",
                "Invalid URL format");
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ConverterException(msg);
        }

        UrlData urlData = new UrlData(url.toString());
        return urlData;
    }
}
```

```

    }

    @Override
    public String getAsString(FacesContext facesContext,
        UIComponent component, Object value) {
        return value.toString();
    }
}

```

### **UserData.java**

```

package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public UrlData data;

    public UrlData getData() {
        return data;
    }

    public void setData(UrlData data) {
        this.data = data;
    }
}

```

**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>Custom Converter Example</h2>
    <h:form>
      <h:inputText id="urlInput" value="#{userData.data}"
        label="URL" >
        <f:converter converterId="com.tutorialspoint.test.UrlConverter" />
      </h:inputText>
      <h:commandButton value="submit" action="result"/>
      <h:message for="urlInput" style="color:red" />
    </h:form>
  </h:body>
</html>

```

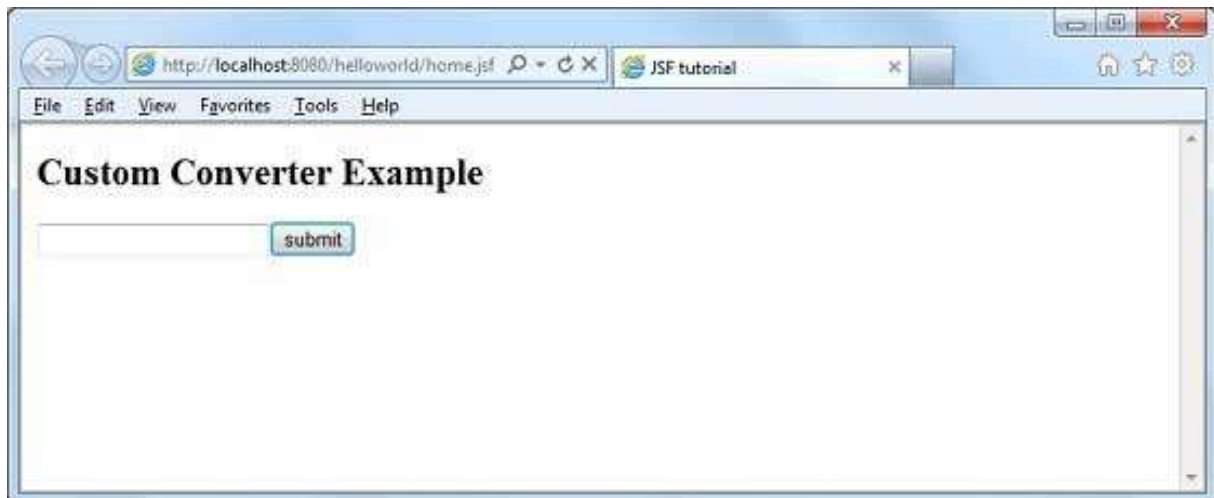
**result.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:body>
    <h2>Result</h2>
    <hr />
    #{userData.data}
  </h:body>
</html>

```

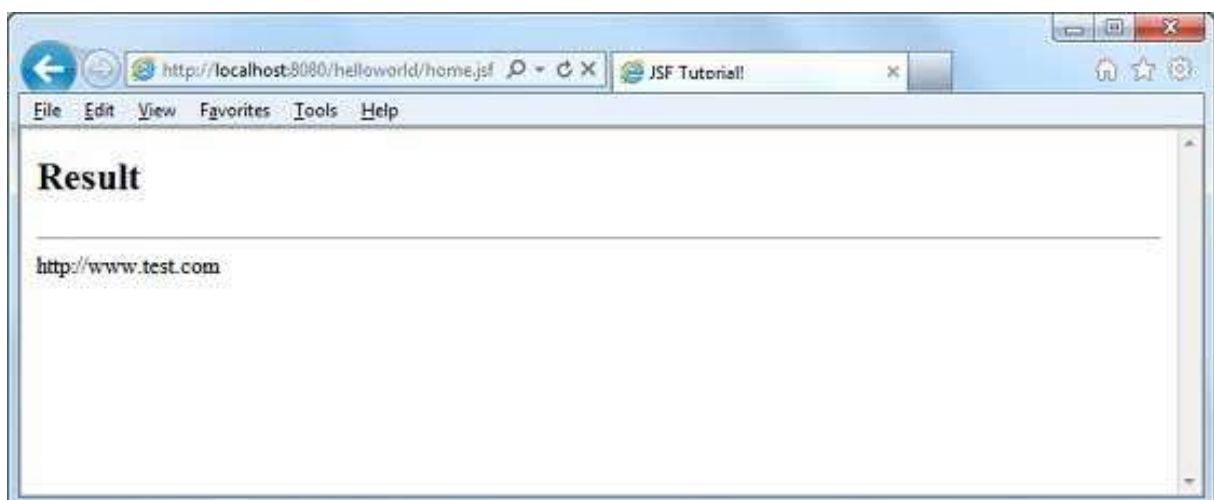
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Enter any invalid value and press Submit button. See the following error message.



Enter any valid value and press Submit button. See the following result.



# 11. JSF - Validator Tags

JSF provides inbuilt validators to validate its UI components. These tags can validate the length of the field, the 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 2.0:

Sr. No.	Tag & Description
1	<b><u>f:validateLength</u></b> Validates the length of a string
2	<b><u>f:validateLongRange</u></b> Validates the range of a numeric value
3	<b><u>f:validateDoubleRange</u></b> Validates the range of a float value
4	<b><u>f:validateRegex</u></b> Validates JSF component with a given regular expression
5	<b><u>Custom Validator</u></b> Creates a custom validator

## f:validateLength

f:validateLength tag is used to validate the length of a string value in a particular range.

### JSF Tag

```
<f:validateLength minimum="5" maximum="8" />
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>minimum</b> A String with a minimum number of characters
2	<b>maximum</b> A String with a maximum number of characters

### Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.

4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure the business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### **UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```



**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>h:validateLength Example</h2>
    <h:form>
      <h:inputText id="nameInput" value="#{userData.name}"
        label="name" >
        <f:validateLength minimum="5" maximum="8" />
      </h:inputText>
      <h:commandButton value="submit" action="result"/>
      <h:message for="nameInput" style="color:red" />
    </h:form>
  </h:body>
</html>

```

**result.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>JSF Tutorial!</title>
  </h:head>
  <h:body>
    <h2>Result</h2>
    <hr />

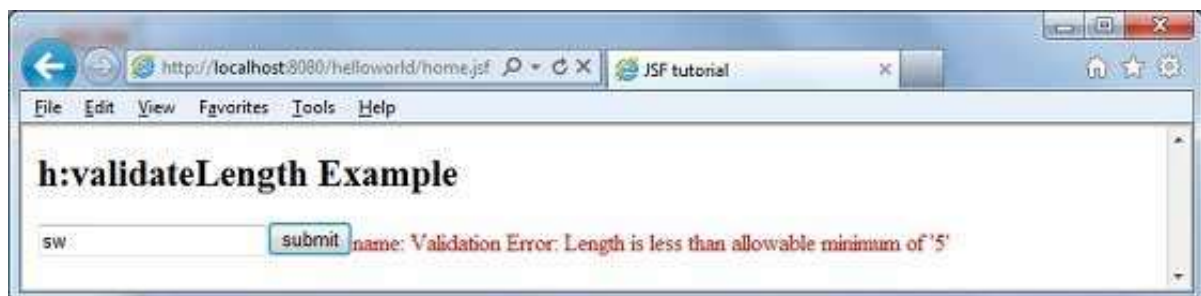
```

```
Name: #{userData.name}  
</h:body>  
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Enter an invalid value. Following will be the output.



Enter a valid value. Following will be the output.



## f:validateLongRange

f:validateLongRange tag is used to validate the long value in a particular range.

### JSF Tag

```
<f:validateLongRange minimum="5" maximum="200" />
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>minimum</b> Minimum long value within an optional range
2	<b>maximum</b> Maximum long value within an optional range

### Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.

4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure the business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### UserData.java

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>h:validateLongRange Example</h2>
    <h:form>
      <h:inputText id="ageInput" value="#{userData.age}"
        label="age" >
        <f:validateLongRange minimum="5" maximum="200" />
      </h:inputText>
      <h:commandButton value="submit" action="result"/>
      <h:message for="ageInput" style="color:red" />
    </h:form>
  </h:body>
</html>

```

**result.xhtml**

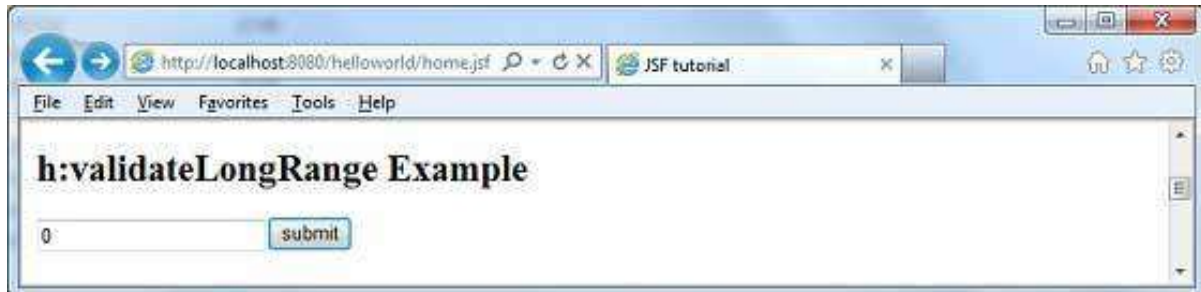
```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>JSF Tutorial!</title>
  </h:head>
  <h:body>
    <h2>Result</h2>
    Age:  #{userData.age}

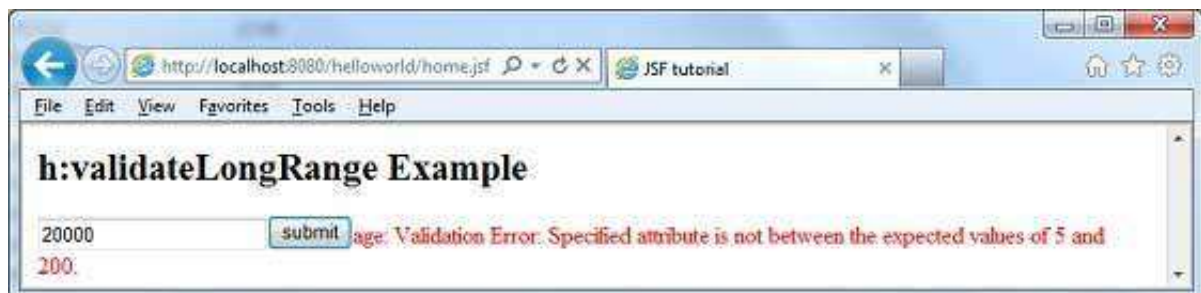
```

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

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Enter an invalid value. Following will be the output.



Enter a valid value. Following will be the output.



## f:validateDoubleRange

f:validateDoubleRange tag is used to validate a value to a range of float values.

### JSF Tag

```
<f:validateDoubleRange minimum="1000.50" maximum="10000.50" />
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>minimum</b> Minimum double value within an optional range
2	<b>maximum</b> Maximum double value within an optional range

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.

4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure the business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### **UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private double salary;

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

}
```



**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>h:validateDoubleRange Example</h2>
    <h:form>
      <h:inputText id="salaryInput" value="#{userData.salary}"
        label="salary" >
        <f:validateDoubleRange minimum="1000.50" maximum="10000.50" />
      </h:inputText>
      <h:commandButton value="submit" action="result"/>
      <h:message for="salaryInput" style="color:red" />
    </h:form>
  </h:body>
</html>

```

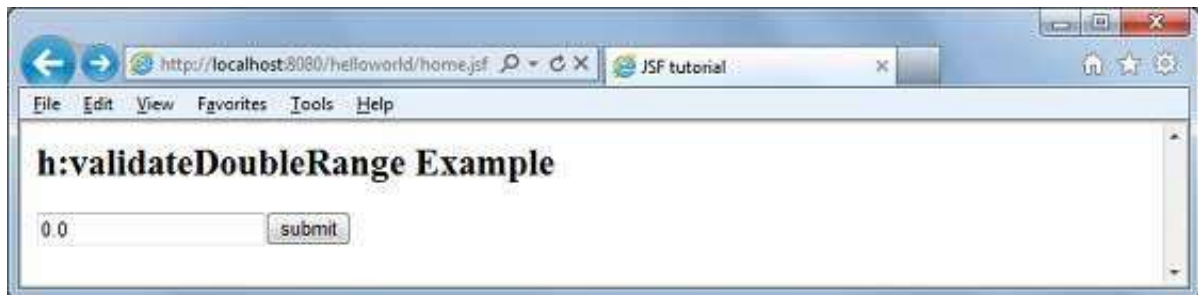
**result.xhtml**

```

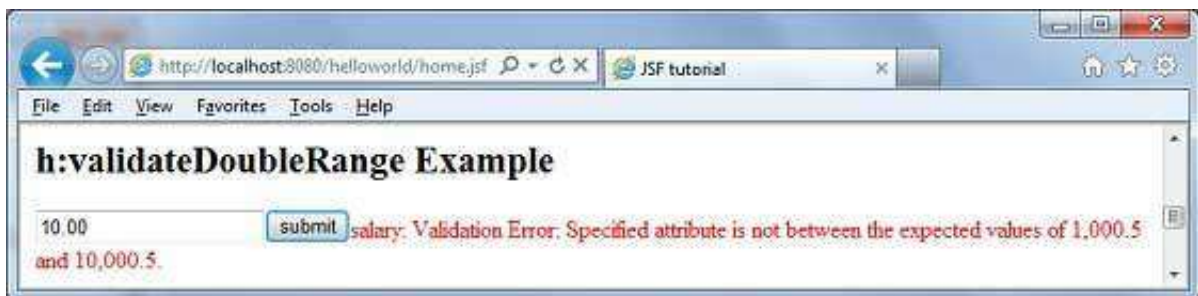
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>JSF Tutorial!</title>
  </h:head>
  <h:body>
    <h2>Result</h2>
    Salary: #{userData.salary}
  </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Enter an invalid value. Following will be the output.



Enter a valid value. Following will be the output.



## f:validateRegex

f:validateRegex tag is used to validate a string value to a required format.

### JSF Tag

```
<f:validateRegex pattern="((?=.*[a-z]).{6,})" />
```

### Tag Attributes

Sr. No.	Attribute & Description
1	<b>pattern</b> Formatting pattern

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Create <i>result.xhtml</i> in the webapps directory as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Compile and run the application to make sure the business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```

package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private String password;
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>h:validateRegex Example</h2>
        <!-- password contains lower case letters only and.
        length of the password should be greater than 6. -->

```

```

<h:form>

    <h:inputSecret id="passwordInput" value="#{userData.password}"

        label="password" >
        <f:validateRegex pattern="((?=.*[a-z]).{6,})" />
    </h:inputSecret>
    <h:commandButton value="submit" action="result"/>
    <h:message for="passwordInput" style="color:red" />
</h:form>
</h:body>
</html>

```

### result.xhtml

```

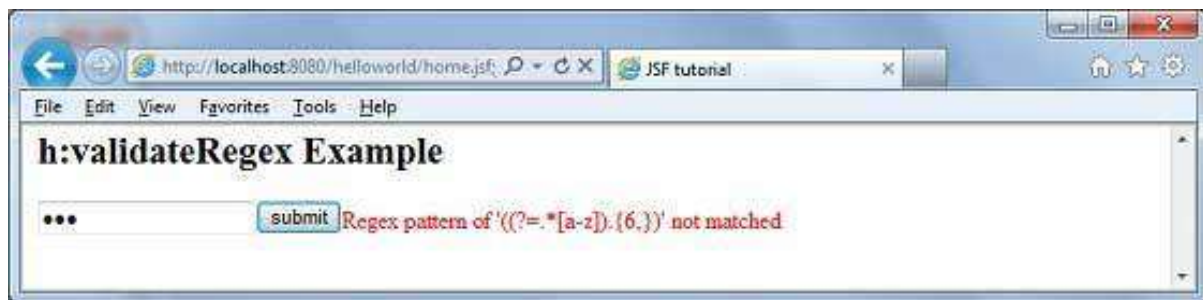
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>JSF Tutorial!</title>
    </h:head>
    <h:body>
        <h2>Result</h2>
        <hr />
        Password: #{userData.password}
    </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Enter an invalid value. Following will be the output.



Enter a valid value. Following will be the output.



## Custom Validator

We can create our own Custom validator in JSF.

Defining a custom validator in JSF is a three-step process.

Step No.	Description
1	Create a validator class by implementing <i>javax.faces.validator.Validator</i> interface.
2	Implement <i>validate()</i> method of the above interface.
3	Use Annotation <i>@FacesValidator</i> to assign a unique ID to the custom validator.

### Step 1: Create a Validator Class : UrlValidator.java

```
public class UrlValidator implements Validator {
    ...
}
```

```
}
```

## Step 2: Implement Validator Interface Methods : UrlValidator.java

```
public class UrlValidator implements Validator {
    @Override
    public void validate(FacesContext facesContext,
        UIComponent component, String value) throws ValidatorException {
        ...
    }
}
```

## Step 3: Annotate to Register the Validator : UrlValidator.java

```
@FacesValidator("com.tutorialspoint.test.UrlValidator")
public class UrlValidator implements Validator {
}
```

## Use the validator in JSF page

```
<h:inputText id="urlInput" value="#{userData.data}" label="URL" >
    <f:validator validatorId="com.tutorialspoint.test.UrlValidator" />
</h:inputText>
```

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Create <i>UrlValidator.java</i> as a converter under package <i>com.tutorialspoint.test</i> as explained below.

3	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
4	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
5	Create <i>result.xhtml</i> in the webapps directory as explained below.
6	Compile and run the application to make sure the business logic is working as per the requirements.
7	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
8	Launch your web application using appropriate URL as explained below in the last step.

### UrlValidator.java

```
package com.tutorialspoint.test;

import java.net.URI;
import java.net.URISyntaxException;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

@FacesValidator("com.tutorialspoint.test.UrlValidator")
public class UrlValidator implements Validator {

    @Override
```



```

public void validate(FacesContext facesContext,
    UIComponent component, Object value)
    throws ValidatorException {

    StringBuilder url = new StringBuilder();
    String urlValue = value.toString();

    if(!urlValue.startsWith("http://", 0)){
        url.append("http://");
    }
    url.append(urlValue);

    try {
        new URI(url.toString());
    } catch (URISyntaxException e) {
        FacesMessage msg =
            new FacesMessage("URL validation failed","Invalid URL format");
        msg.setSeverity(FacesMessage.SEVERITY_ERROR);
        throw new ValidatorException(msg);
    }
}

```

### UserData.java

```

package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    public String data;
}

```

```

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}

```

### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>Custom Validator Example</h2>
        <h:form>
            <h:inputText id="urlInput" value="#{userData.data}"
                label="URL" >
                <f:validator validatorId="com.tutorialspoint.test.UrlValidator" />
            </h:inputText>
            <h:commandButton value="submit" action="result"/>
            <h:message for="urlInput" style="color:red" />
        </h:form>
    </h:body>
</html>

```

### result.xhtml

```

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

```

```

xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets">

<h:body>

    <h2>Result</h2>
    <hr />
    #{userData.data}
</h:body>
</html>

```

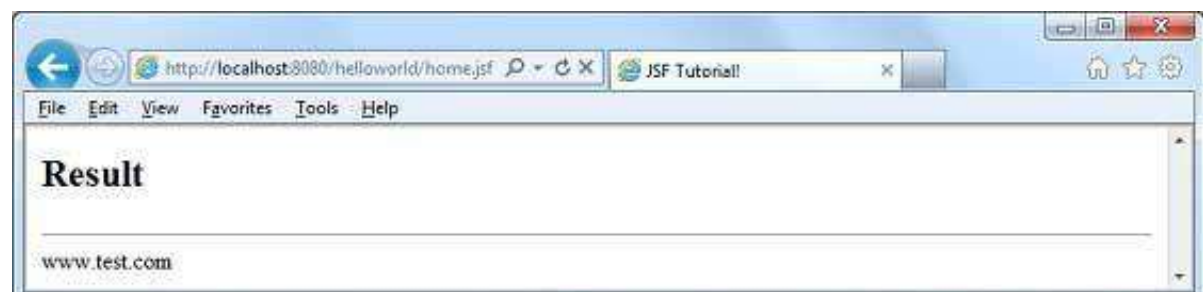
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Enter any invalid value and press Submit button. See the following error message.



Enter any valid value and press Submit button. Following will be the output.





# 12. JSF – DataTable

JSF provides a rich control named DataTable to render and format html tables.

- DataTable can iterate over a collection or array of values to display data.
- DataTable provides attributes to modify its data in an easy way.

## HTML Header

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html">
</html>
```

Following are important *DataTable* operations in JSF 2.0:

Sr. No.	Tag & Description
1	<b><u>Display DataTable</u></b> How to display a dataTable
2	<b><u>Add data</u></b> How to add a new row in a dataTable
3	<b><u>Edit data</u></b> How to edit a row in a dataTable
4	<b><u>Delete data</u></b> How to delete a row in dataTable
5	<b><u>Using DataModel</u></b> Use DataModel to display row numbers in a dataTable

## Display DataTable

h:dataTable tag is used to display data in a tabular fashion.

### JSF Tag

```
<h:dataTable value="#{userData.employees}" var="employee"
  styleClass="employeeTable"
  headerClass="employeeTableHeader"
  rowClasses="employeeTableOddRow,employeeTableEvenRow">
  <h:column>
    <f:facet name="header">Name</f:facet>
    #{employee.name}
  </h:column>
  <h:column>
    <f:facet name="header">Department</f:facet>
    #{employee.department}
  </h:column>
  <h:column>
    <f:facet name="header">Age</f:facet>
    #{employee.age}
  </h:column>
  <h:column>
    <f:facet name="header">Salary</f:facet>
    #{employee.salary}
  </h:column>
</h:dataTable>
```

### Rendered Output

```
<table class="employeeTable">
<thead><tr>
  <th class="employeeTableHeader" scope="col">Name</th>
  <th class="employeeTableHeader" scope="col">Department</th>
  <th class="employeeTableHeader" scope="col">Age</th>
  <th class="employeeTableHeader" scope="col">Salary</th>
</tr></thead>
<tbody>
<tr class="employeeTableOddRow">
```

```

    <td>John</td>
    <td>Marketing</td>
    <td>30</td>
    <td>2000.0</td>
</tr>
<tr class="employeeTableEvenRow">
    <td>Robert</td>
    <td>Marketing</td>
    <td>35</td>
    <td>3000.0</td>
</tr>
</table>

```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>id</b> Identifier for a component
2	<b>rendered</b> A boolean; false suppresses rendering
3	<b>dir</b> Direction for text. Valid values are <b>ltr</b> (left to right) and <b>rtl</b> (right to left)
4	<b>styleClass</b> Cascading stylesheet (CSS) class name
5	<b>value</b> A component's value, typically a value binding

6	<b>bgcolor</b> Background color for the table
7	<b>border</b> Width of the table's border
8	<b>cellpadding</b> Padding around table cells
9	<b>cellspacing</b> Spacing between table cells
10	<b>columnClasses</b> Comma-separated list of CSS classes for columns
11	<b>first</b> Index of the first row shown in the table
12	<b>footerClass</b> CSS class for the table footer
13	<b>frame</b> Specification for sides of the frame surrounding the table should be drawn; valid values: none, above, below, hside, vside, lhs, rhs, box, border
14	<b>headerClass</b> CSS class for the table header
15	<b>rowClasses</b> Comma-separated list of CSS classes for rows
16	<b>rules</b> Specification for lines drawn between cells; valid values: groups, rows, columns, all
17	<b>summary</b> Summary of the table's purpose and structure used for non-visual feedback such as speech



18	<b>var</b> The name of the variable created by the data table that represents the current item in the value
19	<b>title</b> A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value
20	<b>width</b> Width of an element
21	<b>onblur</b> Element loses focus
22	<b>onchange</b> Element's value changes
23	<b>onclick</b> Mouse button is clicked over the element
24	<b>ondblclick</b> Mouse button is double-clicked over the element
25	<b>onfocus</b> Element receives focus
26	<b>onkeydown</b> Key is pressed

27	<b>onkeypress</b> Key is pressed and subsequently released
28	<b>onkeyup</b> Key is released
29	<b>onmousedown</b> Mouse button is pressed over the element
30	<b>onmousemove</b> Mouse moves over the element
31	<b>onmouseout</b> Mouse leaves the element's area
32	<b>onmouseover</b> Mouse moves onto an element
33	<b>onmouseup</b> Mouse button is released

## Example Application

Let us create a test JSF application to test the above tag.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - h:outputStylesheet</i> sub-chapter of <i>JSF - Basic Tags</i> chapter.
2	Modify <i>styles.css</i> as explained below.
3	Create <i>Employee.java</i> under package <i>com.tutorialspoint.test</i> as explained below.
4	Create <i>UserData.java</i> as a managed bean under package <i>com.tutorialspoint.test</i> as explained below.
5	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.

6	Compile and run the application to make sure the business logic is working as per the requirements.
7	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
8	Launch your web application using appropriate URL as explained below in the last step.

**styles.css**

```
.employeeTable{
    border-collapse:collapse;
    border:1px solid #000000;
}

.employeeTableHeader{
    text-align:center;
    background:none repeat scroll 0 0 #B5B5B5;
    border-bottom:1px solid #000000;
    padding:2px;
}

.employeeTableOddRow{
    text-align:center;
    background:none repeat scroll 0 0 #FFFFFFF;
}

.employeeTableEvenRow{
    text-align:center;
    background:none repeat scroll 0 0 #D3D3D3;
}
```

**Employee.java**

```
package com.tutorialspoint.test;

public class Employee {
    private String name;
    private String department;
```

```
private int age;
private double salary;
private boolean canEdit;

public Employee (String name,String department,int age,double salary){
    this.name = name;
    this.department = department;
    this.age = age;
    this.salary = salary;
    canEdit = false;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getDepartment() {
    return department;
}

public void setDepartment(String department) {
    this.department = department;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public double getSalary() {
    return salary;
}
```

```

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public boolean isCanEdit() {
        return canEdit;
    }

    public void setCanEdit(boolean canEdit) {
        this.canEdit = canEdit;
    }
}

```

### **UserData.java**

```

package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Arrays;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private String name;
    private String dept;
    private int age;
    private double salary;

    private static final ArrayList<Employee> employees

```

```

    = new ArrayList<Employee>(Arrays.asList(
        new Employee("John", "Marketing", 30,2000.00),
        new Employee("Robert", "Marketing", 35,3000.00),
        new Employee("Mark", "Sales", 25,2500.00),
        new Employee("Chris", "Marketing", 33,2500.00),
        new Employee("Peter", "Customer Care", 20,1500.00)
    ));

    public ArrayList<Employee> getEmployees() {
        return employees;
    }

    public String addEmployee() {
        Employee employee = new Employee(name,dept,age,salary);
        employees.add(employee);
        return null;
    }

    public String deleteEmployee(Employee employee) {
        employees.remove(employee);
        return null;
    }

    public String editEmployee(Employee employee){
        employee.setCanEdit(true);
        return null;
    }

    public String saveEmployees(){
        //set "canEdit" of all employees to false
        for (Employee employee : employees){
            employee.setCanEdit(false);
        }
        return null;
    }

    public String getName() {

```

```
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDepartment() {
        return department;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

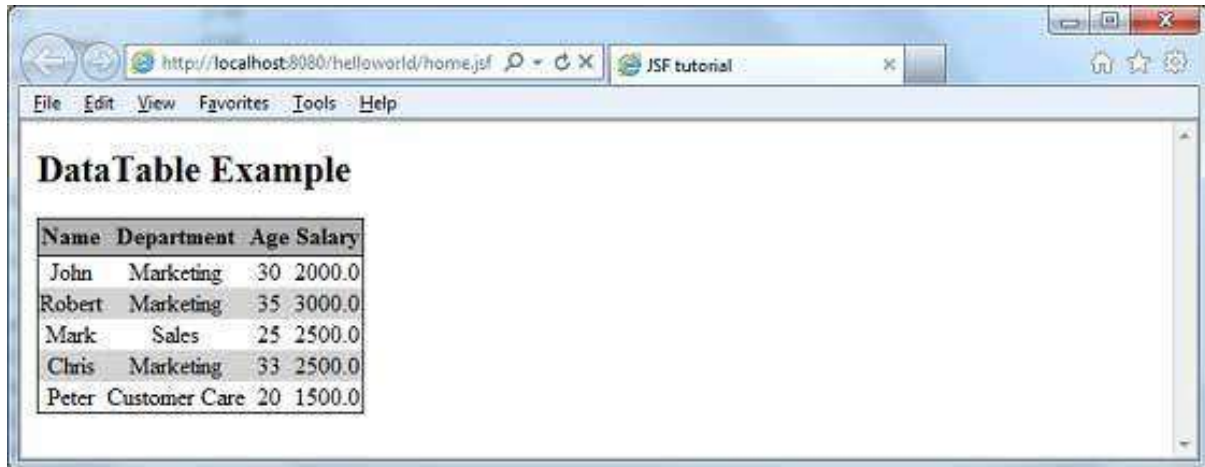
### home.xhtml

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

```
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
    <h:outputStylesheet library="css" name="styles.css" />
  </h:head>
  <h:body>
    <h2>DataTable Example</h2>
    <h:form>
      <h:dataTable value="#{userData.employees}" var="employee"
        styleClass="employeeTable"
        headerClass="employeeTableHeader"
        rowClasses="employeeTableOddRow,employeeTableEvenRow">
        <h:column>
          <f:facet name="header">Name</f:facet>
          #{employee.name}
        </h:column>
        <h:column>
          <f:facet name="header">Department</f:facet>
          #{employee.department}
        </h:column>
        <h:column>
          <f:facet name="header">Age</f:facet>
          #{employee.age}
        </h:column>
        <h:column>
          <f:facet name="header">Salary</f:facet>
          #{employee.salary}
        </h:column>
      </h:dataTable>
    </h:form>
  </h:body>
</html>
```



Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



## Add Data to DataTable

In this section, we'll showcase adding a row to a dataTable.

### Example Application

Let us create a test JSF application to test the above functionality.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - Display DataTable</i> sub-chapter of <i>JSF - DataTables</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Compile and run the application to make sure the business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

5	Launch your web application using appropriate URL as explained below in the last step.
---	--

### home.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
    <h:outputStylesheet library="css" name="styles.css" />
  </h:head>
  <h:body>
    <h2>DataTable Example</h2>
    <h:form>
      <h:dataTable value="#{userData.employees}" var="employee"
        styleClass="employeeTable"
        headerClass="employeeTableHeader"
        rowClasses="employeeTableOddRow,employeeTableEvenRow">
        <h:column>
          <f:facet name="header">Name</f:facet>
          #{employee.name}
        </h:column>
        <h:column>
          <f:facet name="header">Department</f:facet>
          #{employee.department}
        </h:column>
        <h:column>
          <f:facet name="header">Age</f:facet>
          #{employee.age}
        </h:column>
        <h:column>
          <f:facet name="header">Salary</f:facet>

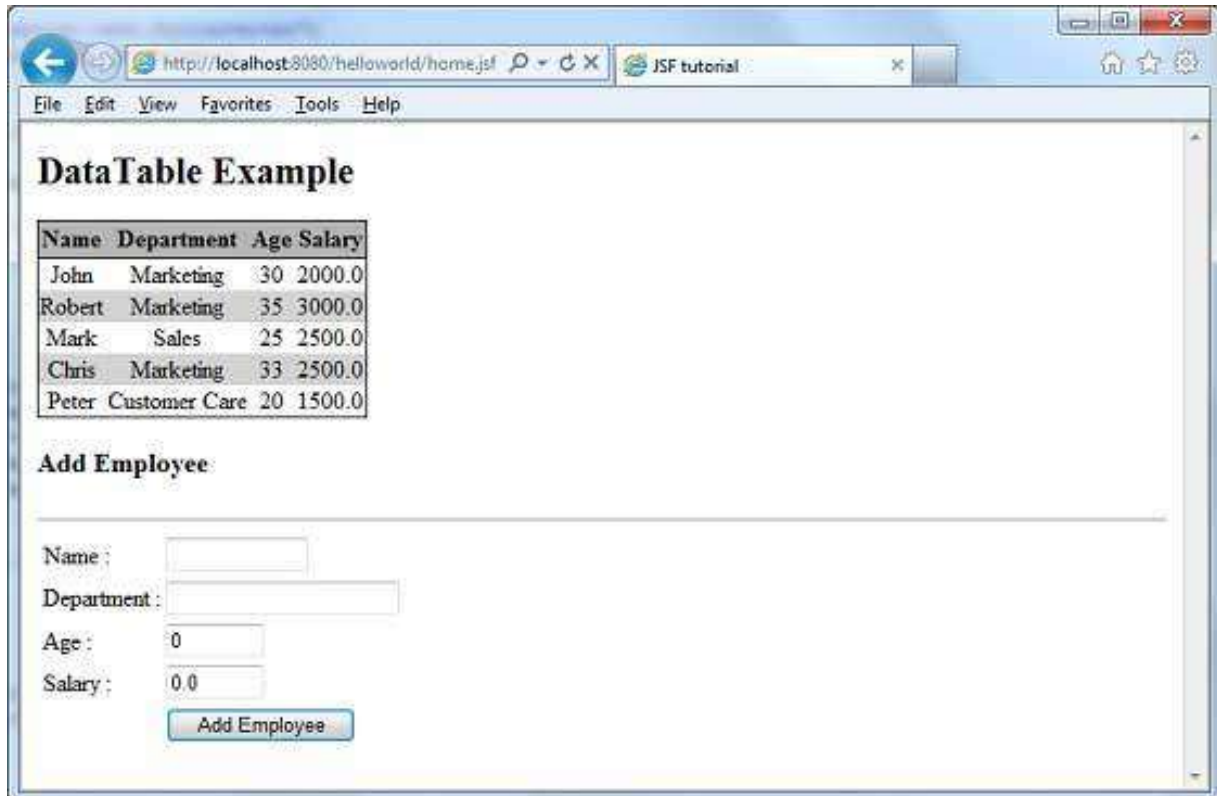
```

```

        #{employee.salary}
    </h:column>
</h:dataTable>
<h3>Add Employee</h3>
<hr/>
<table>
<tr>
    <td>Name :</td>
    <td><h:inputText size="10" value="#{userData.name}" /></td>
</tr>
<tr>
    <td>Department :</td>
    <td><h:inputText size="20" value="#{userData.dept}" /></td>
</tr>
<tr>
    <td>Age :</td>
    <td><h:inputText size="5" value="#{userData.age}" /></td>
</tr>
<tr>
    <td>Salary :</td>
    <td><h:inputText size="5" value="#{userData.salary}" /></td>
</tr>
<tr>
    <td> </td>
    <td><h:commandButton value="Add Employee"
        action="#{userData.addEmployee}" /></td>
</tr>
</table>
</h:form>
</h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



**DataTable Example**

Name	Department	Age	Salary
John	Marketing	30	2000.0
Robert	Marketing	35	3000.0
Mark	Sales	25	2500.0
Chris	Marketing	33	2500.0
Peter	Customer Care	20	1500.0

**Add Employee**

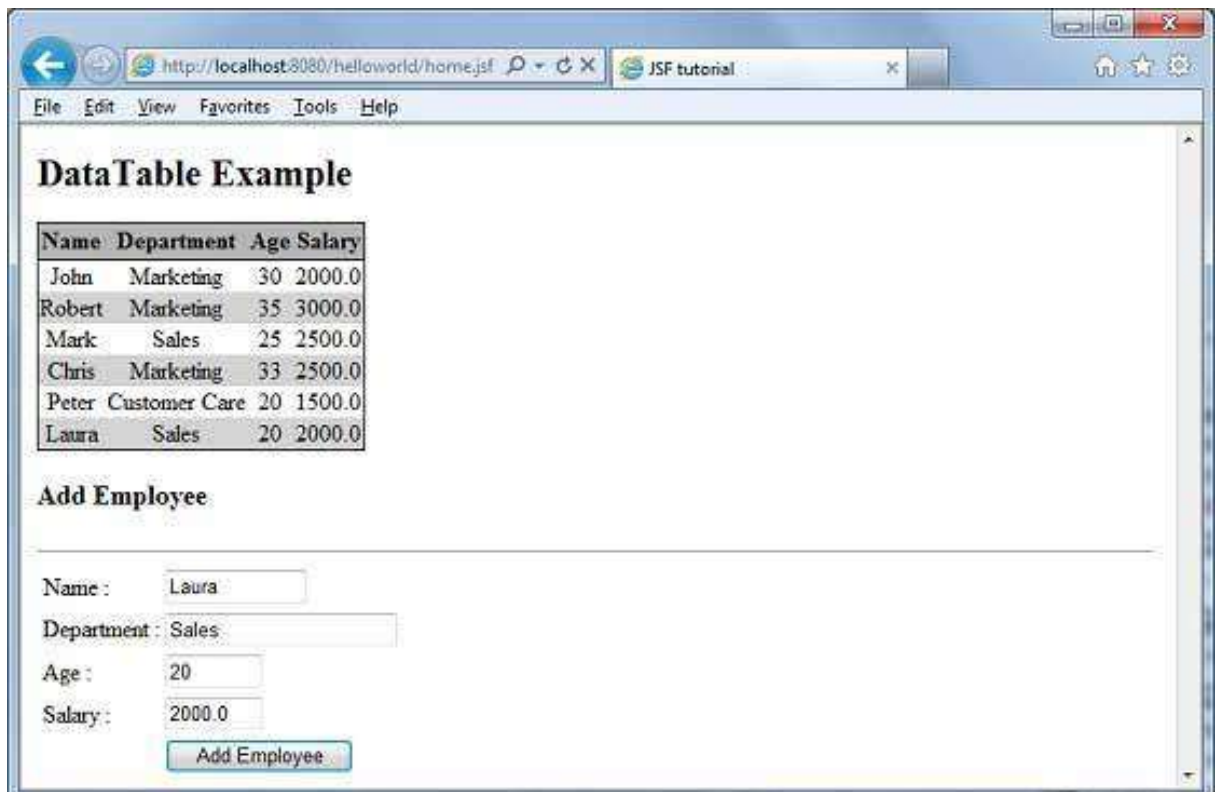
Name :

Department :

Age :

Salary :

Add values to *Add Employee* Form and click *Add Employee* button. See the following result.



**DataTable Example**

Name	Department	Age	Salary
John	Marketing	30	2000.0
Robert	Marketing	35	3000.0
Mark	Sales	25	2500.0
Chris	Marketing	33	2500.0
Peter	Customer Care	20	1500.0
Laura	Sales	20	2000.0

**Add Employee**

Name :

Department :

Age :

Salary :

## Edit Data of a DataTable

In this section, we'll showcase the adding editing capability to a row in a dataTable.

### Example Application

Let us create a test JSF application to test the above functionality.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - Display DataTable</i> sub-chapter of <i>JSF - Data Tables</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Compile and run the application to make sure the business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

#### home.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>JSF tutorial</title>
        <h:outputStylesheet library="css" name="styles.css" />
    </h:head>
    <h:body>
        <h2>DataTable Example</h2>
        <h:form>
            <h:dataTable value="#{userData.employees}" var="employee"
                styleClass="employeeTable"
                headerClass="employeeTableHeader"
                rowClasses="employeeTableOddRow,employeeTableEvenRow">
                <h:column>
```

```

        <f:facet name="header">Name</f:facet>
        <h:inputText value="#{employee.name}"
            size="10" rendered="#{employee.canEdit}" />
        <h:outputText value="#{employee.name}"
            rendered="#{not employee.canEdit}" />
    </h:column>
    <h:column>
        <f:facet name="header">Department</f:facet>
        <h:inputText value="#{employee.department}"
            size="20" rendered="#{employee.canEdit}" />
        <h:outputText value="#{employee.department}"
            rendered="#{not employee.canEdit}" />
    </h:column>
    <h:column>
        <f:facet name="header">Age</f:facet>
        <h:inputText value="#{employee.age}" size="5"
            rendered="#{employee.canEdit}" />
        <h:outputText value="#{employee.age}"
            rendered="#{not employee.canEdit}" />
    </h:column>
    <h:column>
        <f:facet name="header">Salary</f:facet>
        <h:inputText value="#{employee.salary}"
            size="5" rendered="#{employee.canEdit}" />
        <h:outputText value="#{employee.salary}"
            rendered="#{not employee.canEdit}" />
    </h:column>
    <h:column>
        <f:facet name="header">Edit</f:facet>
        <h:commandButton value="Edit"
            action="#{userData.editEmployee}"
            rendered="#{not employee.canEdit}">
            <f:setPropertyActionListener
                target="#{userData.employee}" value="#{employee}" />
        </h:commandButton>
    </h:column>
</h:dataTable>
<br/>

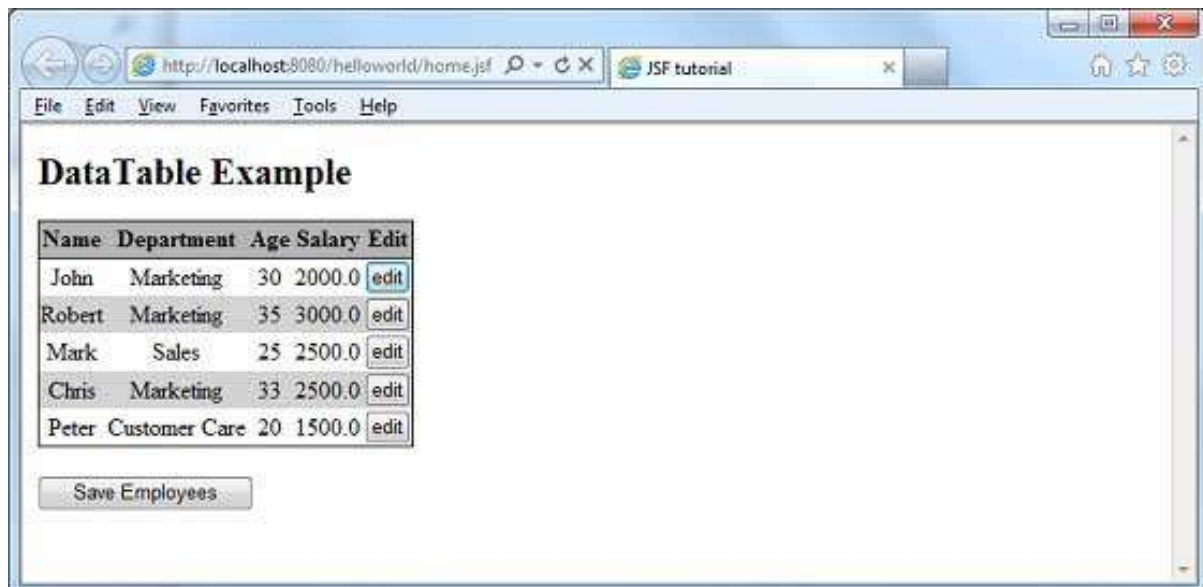
```

```

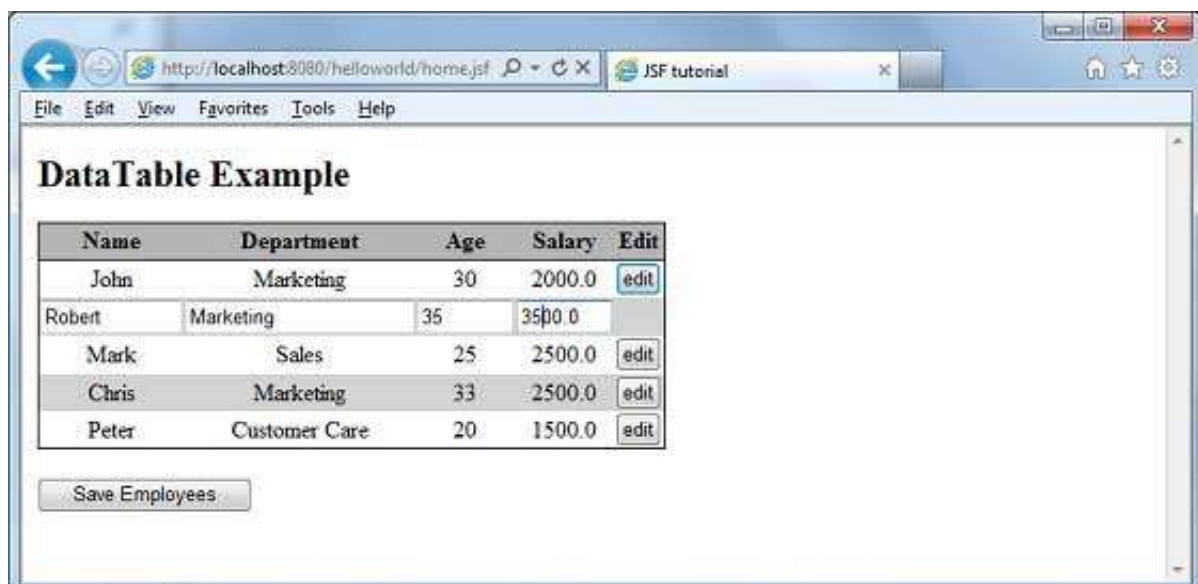
<h:commandButton value="Save Employees"
    action="#{userData.saveEmployees}" />
</h:form>
</h:body>
</html>

```

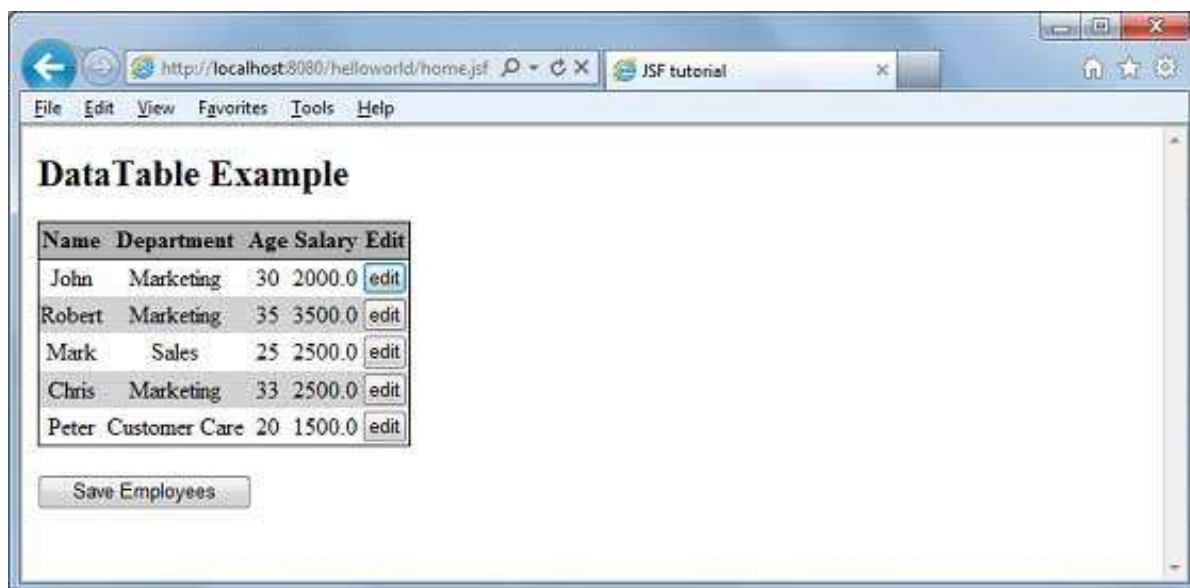
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Click the *edit* button of any row. Following will be the output.



Click *Save Employees* button to save the edit. Following will be the output



## Delete Data of a DataTable

In this section, we'll showcase the adding deleting capability in dataTable.

### Example Application

Let us create a test JSF application to test the above functionality.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - Display DataTable</i> sub-chapter of <i>JSF - DataTables</i> chapter.
2	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
3	Compile and run the application to make sure the business logic is working as per the requirements.
4	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.



**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>JSF tutorial</title>
        <h:outputStylesheet library="css" name="styles.css" />
    </h:head>
    <h:body>
        <h2>DataTable Example</h2>
        <h:form>
            <h:dataTable value="#{userData.employees}" var="employee"
                styleClass="employeeTable"
                headerClass="employeeTableHeader"
                rowClasses="employeeTableOddRow,employeeTableEvenRow">
                <h:column>
                    <f:facet name="header">Name</f:facet>
                    <h:inputText value="#{employee.name}"
                        size="10" rendered="#{employee.canEdit}" />
                    <h:outputText value="#{employee.name}"
                        rendered="#{not employee.canEdit}" />
                </h:column>
                <h:column>
                    <f:facet name="header">Department</f:facet>
                    <h:inputText value="#{employee.department}"
                        size="20" rendered="#{employee.canEdit}" />
                    <h:outputText value="#{employee.department}"
                        rendered="#{not employee.canEdit}" />
                </h:column>
                <h:column>
                    <f:facet name="header">Age</f:facet>
                    <h:inputText value="#{employee.age}" size="5"
                        rendered="#{employee.canEdit}" />
                    <h:outputText value="#{employee.age}"
                        rendered="#{not employee.canEdit}" />
                </h:column>
            </h:dataTable>
        </h:form>
    </h:body>
</html>

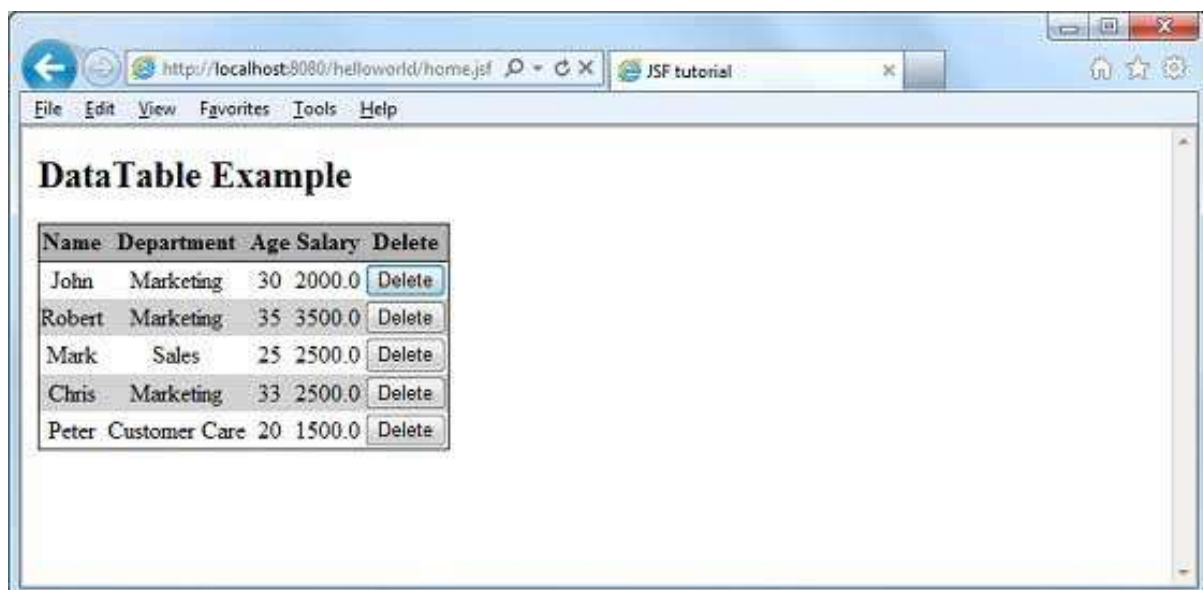
```

```

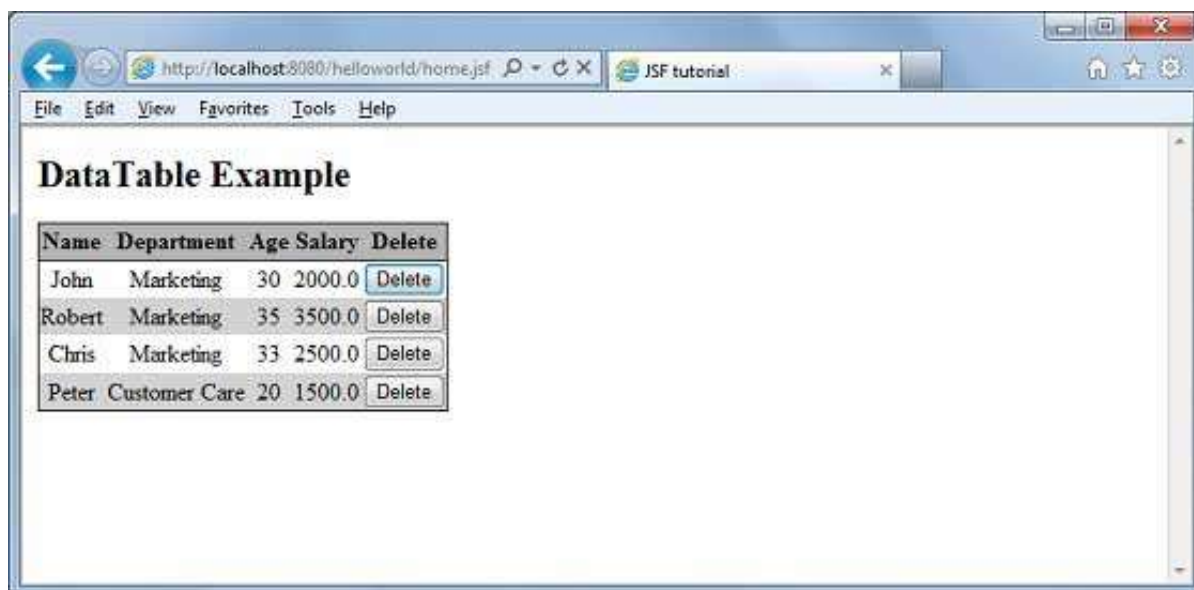
</h:column>
<h:column>
    <f:facet name="header">Salary</f:facet>
    <h:inputText value="#{employee.salary}"
        size="5" rendered="#{employee.canEdit}" />
    <h:outputText value="#{employee.salary}"
        rendered="#{not employee.canEdit}" />
</h:column>
<h:column>
    <f:facet name="header">Delete</f:facet>
    <h:commandButton value="Delete"
        action="#{userData.deleteEmployee}" />
    <f:setPropertyActionListener
        target="#{userData.employee}" value="#{employee}" />
</h:commandButton>
</h:column>
</h:dataTable>
</h:form>
</h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Click *delete* button of any row. Following will be the output.



## Using DataModel in a DataTable

In this section, we'll showcase the use of `datamodel` in a `dataTable`.

### Example Application

Let us create a test JSF application to test the above functionality.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - Display DataTable</i> sub-chapter of <i>JSF - DataTables</i> chapter.
2	Modify <i>UserData.java</i> as explained below.
3	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.

4	Compile and run the application to make sure the business logic is working as per the requirements.
5	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
6	Launch your web application using appropriate URL as explained below in the last step.

### **UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.model.ArrayDataModel;
import javax.faces.model.DataModel;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private static final Employee[] employees = new Employee[] {
        new Employee("John", "Marketing", 30,2000.00),
        new Employee("Robert", "Marketing", 35,3000.00),
        new Employee("Mark", "Sales", 25,2500.00),
        new Employee("Chris", "Marketing", 33,2500.00),
        new Employee("Peter", "Customer Care", 20,1500.00)
    };

    private DataModel<Employee> employeeDataModel
```

```

    = new ArrayDataModel<Employee>(employees);

    public DataModel<Employee> getEmployees() {
        return employeeDataModel;
    }
}

```

## home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>JSF tutorial</title>
        <h:outputStylesheet library="css" name="styles.css" />
    </h:head>
    <h:body>
        <h2>DataTable Example</h2>
        <h:form>
            <h:dataTable value="#{userData.employees}" var="employee"
                styleClass="employeeTable"
                headerClass="employeeTableHeader"
                rowClasses="employeeTableOddRow,employeeTableEvenRow">
                <h:column>
                    <f:facet name="header">Sr. No</f:facet>
                    #{userData.employees.rowIndex + 1}
                </h:column>
                <h:column>
                    <f:facet name="header">Name</f:facet>
                    #{employee.name}
                </h:column>
                <h:column>
                    <f:facet name="header">Department</f:facet>
                    #{employee.department}

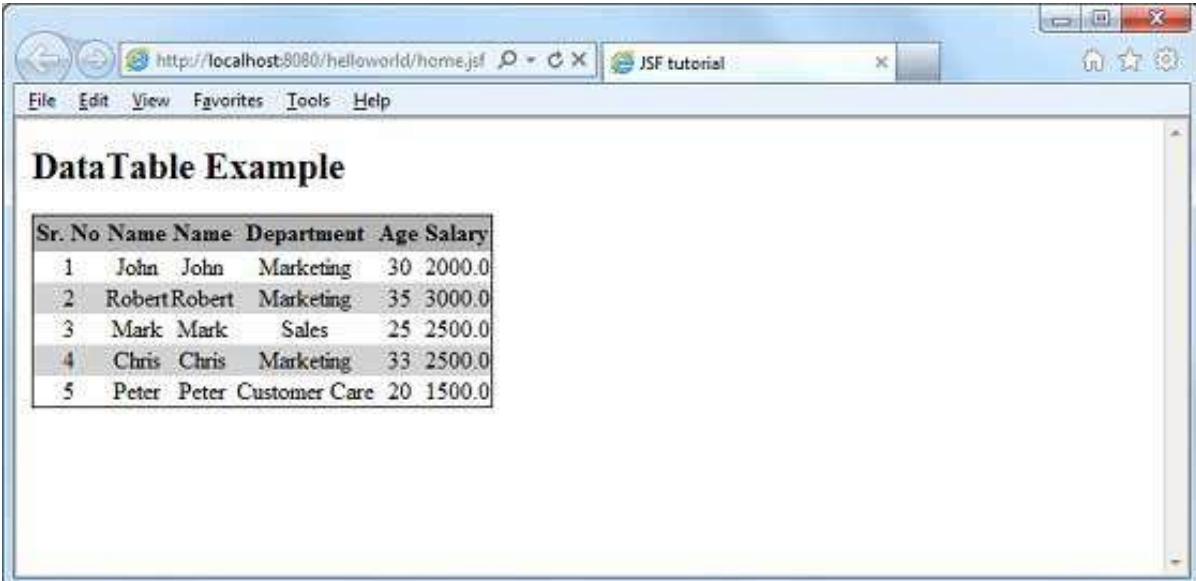
```

```

</h:column>
<h:column>
    <f:facet name="header">Age</f:facet>
    #{employee.age}
</h:column>
<h:column>
    <f:facet name="header">Salary</f:facet>
    #{employee.salary}
</h:column>
</h:dataTable>
</h:form>
</h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



**DataTable Example**

Sr. No	Name	Name	Department	Age	Salary
1	John	John	Marketing	30	2000.0
2	Robert	Robert	Marketing	35	3000.0
3	Mark	Mark	Sales	25	2500.0
4	Chris	Chris	Marketing	33	2500.0
5	Peter	Peter	Customer Care	20	1500.0

# 13. JSF – Composite Components

JSF provides the developers with a powerful capability to define their own custom components, which can be used to render custom contents.

## Define Custom Component

Defining a custom component in JSF is a two-step process.

Step No.	Description
1a	Create a resources folder. Create a xhtml file in resources folder with a composite namespace.
1b	Use composite tags <i>composite:interface</i> , <i>composite:attribute</i> and <i>composite:implementation</i> , to define content of the composite component. Use <i>cc.attrs</i> in <i>composite:implementation</i> to get variable defined using <i>composite:attribute</i> in <i>composite:interface</i> .

### Step 1a: Create Custom Component : loginComponent.xhtml

Create a folder tutorialspoint in resources folder and create a file loginComponent.xhtml in it.

Use composite namespace in html header.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:composite="http://java.sun.com/jsf/composite"
      >
...
</html>
```

## Step 1b: Use Composite Tags : loginComponent.xhtml

Following table describes the use of composite tags.

Sr. No.	Tag & Description
1	<b>composite:interface</b> Declares configurable values to be used in composite:implementation
2	<b>composite:attribute</b> Configuration values are declared using this tag
3	<b>composite:implementation</b> Declares JSF component. Can access the configurable values defined in composite:interface using <code>#{cc.attrs.attribute-name}</code> expression

```

<composite:interface>
    <composite:attribute name="usernameLabel" />
    <composite:attribute name="usernameValue" />
</composite:interface>
<composite:implementation>
<h:form>
    #{cc.attrs.usernameLabel} :
    <h:inputText id="username" value="#{cc.attrs.usernameValue}" />
</h:form>

```



## Use Custom Component

Using a custom component in JSF is a simple process.

Step No.	Description
2a	Create a xhtml file and use custom component's namespace. Namespace will be the <i>http://java.sun.com/jsf/&lt;folder-name&gt;</i> where <i>folder-name</i> is folder in resources directory containing the custom component
2b	Use the custom component as normal JSF tags

### Step 2a: Use Custom Namespace: home.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:tp="http://java.sun.com/jsf/composite/tutorialspoint">
```

### Step 2b: Use Custom Tag: home.xhtml and Pass Values

```
<h:form>
  <tp:loginComponent
    usernameLabel="Enter User Name: "
    usernameValue="#{userData.name}" />
</h:form>
```

## Example Application

Let us create a test JSF application to test the custom component in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Create <i>resources</i> folder under <i>src -&gt; main</i> folder.

3	Create <i>tutorialspoint</i> folder under <i>src -&gt; main -&gt; resources</i> folder.
4	Create <i>loginComponent.xhtml</i> file under <i>src -&gt; main -&gt; resources -&gt; tutorialspoint</i> folder.
5	Modify <i>UserData.java</i> file as explained below.
6	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
7	Compile and run the application to make sure the business logic is working as per the requirements.
8	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
9	Launch your web application using appropriate URL as explained below in the last step.

### loginComponent.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:composite="http://java.sun.com/jsf/composite">
  <composite:interface>
    <composite:attribute name="usernameLabel" />
    <composite:attribute name="usernameValue" />
    <composite:attribute name="passwordLabel" />
    <composite:attribute name="passwordValue" />
```

```

    <composite:attribute name="loginButtonLabel" />
    <composite:attribute name="loginButtonAction"
        method-signature="java.lang.String login()" />
</composite:interface>
<composite:implementation>
    <h:form>
        <h:message for="loginPanel" style="color:red;" />
        <h:panelGrid columns="2" id="loginPanel">
            #{cc.attrs.usernameLabel} :
            <h:inputText id="username" value="#{cc.attrs.usernameValue}" />
            #{cc.attrs.passwordLabel} :
            <h:inputSecret id="password" value="#{cc.attrs.passwordValue}" />
        </h:panelGrid>
        <h:commandButton action="#{cc.attrs.loginButtonAction}"
            value="#{cc.attrs.loginButtonLabel}" />
    </h:form>
</composite:implementation>
</html>

```

### UserData.java

```

package com.tutorialspoint.test;

import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private String name;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {

```

```

        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String login(){
        return "result";
    }
}

```

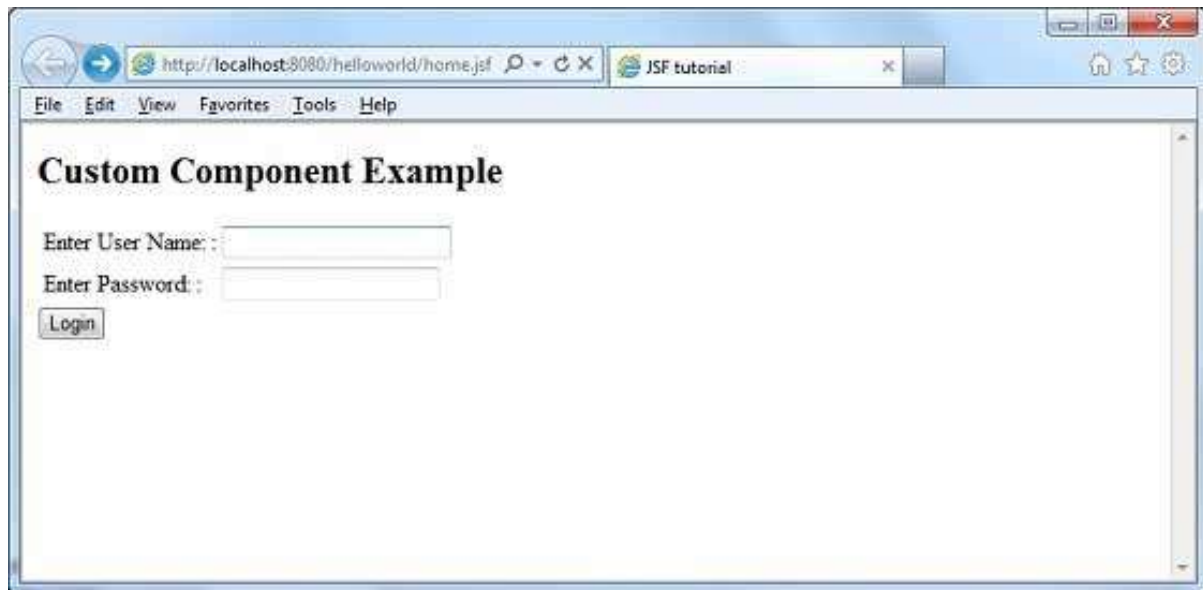
### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:tp="http://java.sun.com/jsf/composite/tutorialspoint">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>Custom Component Example</h2>
        <h:form>
            <tp:loginComponent
                usernameLabel="Enter User Name: "
                usernameValue="#{userData.name}"
                passwordLabel="Enter Password: "
                passwordValue="#{userData.password}"
                loginButtonLabel="Login"
                loginButtonAction="#{userData.login}" />
        </h:form>
    </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



# 14. JSF – Ajax

AJAX stands for Asynchronous JavaScript and Xml.

Ajax is a technique to use XMLHttpRequest of JavaScript to send data to the server and receive data from the server asynchronously. Thus using Ajax technique, javascript code exchanges data with the server, updates parts of the web page without reloading the whole page.

JSF provides excellent support for making ajax call. It provides f:ajax tag to handle ajax calls.

## JSF Tag

```
<f:ajax execute="input-component-name" render="output-component-name" />
```

## Tag Attributes

Sr. No.	Attribute & Description
1	<b>disabled</b> If true, the Ajax behavior will be applied to any parent or child components. If false, the Ajax behavior will be disabled.
2	<b>Event</b> The event that will invoke Ajax requests, for example "click", "change", "blur", "keypress", etc.
3	<b>Execute</b> A space-separated list of IDs for components that should be included in the Ajax request.
4	<b>Immediate</b> If "true" behavior events generated from this behavior are broadcast during Apply Request Values phase. Otherwise, the events will be broadcast during Invoke Applications phase.

5	<b>Listener</b> An EL expression for a method in a backing bean to be called during the Ajax request.
6	<b>Onerror</b> The name of a JavaScript callback function that will be invoked if there is an error during the Ajax request.
7	<b>Onevent</b> The name of a JavaScript callback function that will be invoked to handle UI events.
8	<b>Render</b> A space-separated list of IDs for components that will be updated after an Ajax request.

## Example Application

Let us create a test JSF application to test the custom component in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>UserData.java</i> file as explained below.
3	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.

4	Compile and run the application to make sure the business logic is working as per the requirements.
5	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
6	Launch your web application using appropriate URL as explained below in the last step.

### **UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

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

    public String getWelcomeMessage(){
        return "Hello " + name;
    }
}
```



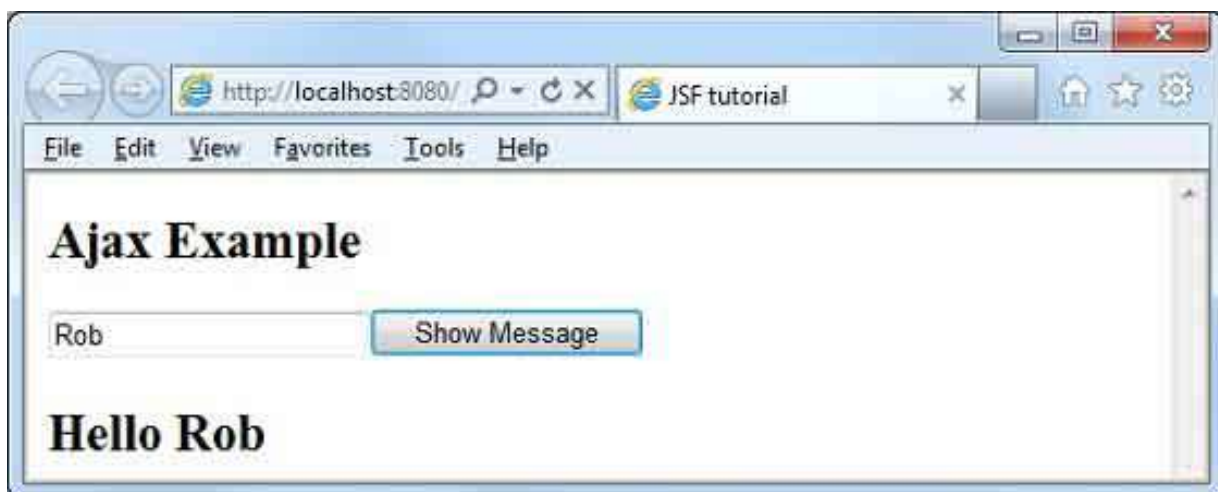
**home.xhtml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:tp="http://java.sun.com/jsf/composite/tutorialspoint">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>Ajax Example</h2>
    <h:form>
      <h:inputText id="inputName" value="#{userData.name}"></h:inputText>
      <h:commandButton value="Show Message">
        <f:ajax execute="inputName" render="outputMessage" />
      </h:commandButton>
      <h2><h:outputText id="outputMessage"
        value="#{userData.welcomeMessage !=null ?
          userData.welcomeMessage : ''}"
        /></h2>
    </h:form>
  </h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Enter the name and press the *Show Message* button. You will see the following result without page refresh/form submit.



# 15. JSF – Event Handling

When a user clicks a JSF button or link or changes any value in the text field, JSF UI component fires an event, which will be handled by the application code. To handle such an event, an event handler is to be registered in the application code or managed bean.

When a UI component checks that a user event has occurred, it creates an instance of the corresponding event class and adds it to an event list. Then, Component fires the event, i.e., checks the list of listeners for that event and calls the event notification method on each listener or handler.

JSF also provide system level event handlers, which can be used to perform some tasks when the application starts or is stopping.

Following are some important *Event Handler* in JSF 2.0:

Sr. No.	Event Handlers & Description
1	<b><u>valueChangeListener</u></b> Value change events get fired when the user make changes in input components.
2	<b><u>actionListener</u></b> Action events get fired when the user clicks a button or link component.
3	<b><u>Application Events</u></b> Events firing during JSF lifecycle: PostConstructApplicationEvent, PreDestroyApplicationEvent , PreRenderViewEvent.

## valueChangeListener

When the user interacts with input components, such as h:inputText or h:selectOneMenu, the JSF fires a valueChangeEvent, which can be handled in two ways.

Technique	Description
<b>Method Binding</b>	Pass the name of the managed bean method in <i>valueChangeListener</i> attribute of UI Component.
<b>ValueChangeListener</b>	Implement ValueChangeListener interface and pass the implementation class name to <i>valueChangeListener</i> attribute of UI Component.

## Method Binding

Define a method

```
public void localeChanged(ValueChangeEvent e){
    //assign new value to country
    selectedCountry = e.getNewValue().toString();
}
```

Use the above method

```
<h:selectOneMenu value="#{userData.selectedCountry}" onchange="submit()"
    valueChangeListener="#{userData.localeChanged}" >
    <f:selectItems value="#{userData.countries}" />
</h:selectOneMenu>
```

## ValueChangeListener

Implement ValueChangeListener

```
public class LocaleChangeListener implements ValueChangeListener {
    @Override
    public void processValueChange(ValueChangeEvent event)
        throws AbortProcessingException {
        //access country bean directly
        UserData userData = (UserData) FacesContext.getCurrentInstance().
            getExternalContext().getSessionMap().get("userData");
        userData.setSelectedCountry(event.getNewValue().toString());
    }
}
```

Use listener method

```
<h:selectOneMenu value="#{userData.selectedCountry}" onchange="submit()">
    <f:valueChangeListener type="com.tutorialspoint.test.LocaleChangeListener"
        />
    <f:selectItems value="#{userData.countries}" />
</h:selectOneMenu>
```

## Example Application

Let us create a test JSF application to test the `valueChangeListener` in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>UserData.java</i> file as explained below.
3	Create <i>LocaleChangeListener.java</i> file under a package <i>com.tutorialspoint.test</i> . Modify it as explained below.
4	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
5	Compile and run the application to make sure the business logic is working as per the requirements.
6	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
7	Launch your web application using appropriate URL as explained below in the last step.

### UserData.java

```
package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.Map;
```

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.event.ValueChangeEvent;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private static Map<String,String> countryMap;
    private String selectedCountry = "United Kingdom"; //default value

    static{
        countryMap = new LinkedHashMap<String,String>();
        countryMap.put("en", "United Kingdom"); //locale, country name
        countryMap.put("fr", "French");
        countryMap.put("de", "German");
    }

    public void localeChanged(ValueChangeEvent e){
        //assign new value to country
        selectedCountry = e.getNewValue().toString();
    }

    public Map<String, String> getCountries() {
        return countryMap;
    }

    public String getSelectedCountry() {
        return selectedCountry;
    }

    public void setSelectedCountry(String selectedCountry) {
        this.selectedCountry = selectedCountry;
    }
}
```

**LocaleChangeListener.java**

```

package com.tutorialspoint.test;

import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ValueChangeEvent;
import javax.faces.event.ValueChangeListener;

public class LocaleChangeListener implements ValueChangeListener {
    @Override
    public void processValueChange(ValueChangeEvent event)
        throws AbortProcessingException {
        //access country bean directly
        UserData userData = (UserData) FacesContext.getCurrentInstance().
            getExternalContext().getSessionMap().get("userData");

        userData.setSelectedCountry(event.getNewValue().toString());
    }
}

```

**home.xhtml**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>valueChangeListener Examples</h2>
        <h:form>
            <h2>Method Binding</h2>

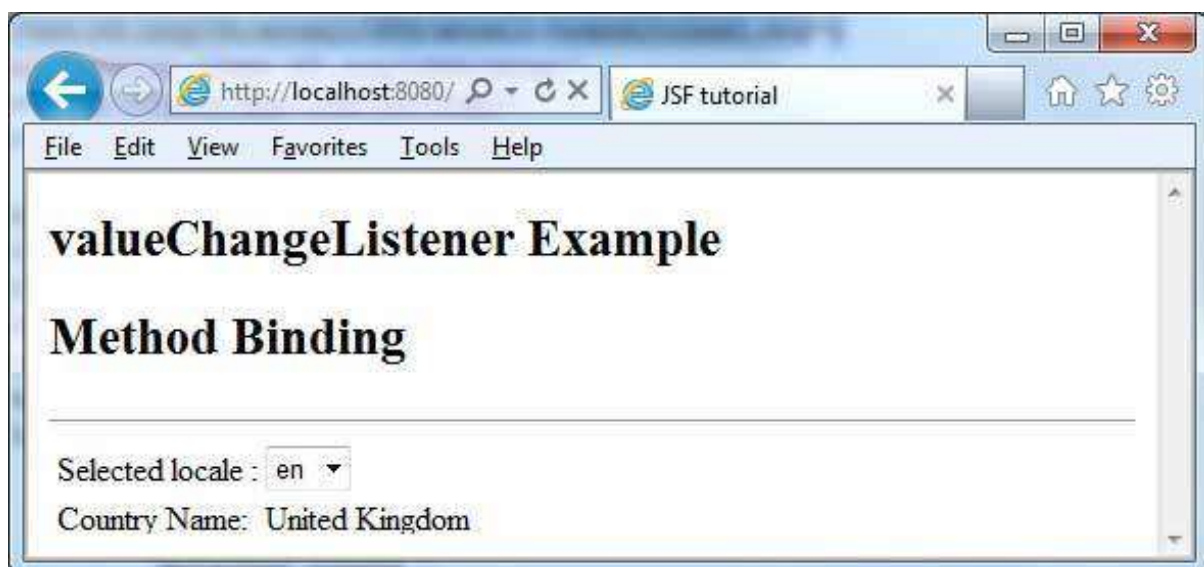
```

```

<hr/>
<h:panelGrid columns="2">
    Selected locale :
    <h:selectOneMenu value="#{userData.selectedCountry}"
        onchange="submit()"
        valueChangeListener="#{userData.localeChanged}" >
        <f:selectItems value="#{userData.countries}" />
    </h:selectOneMenu>
    Country Name:
    <h:outputText id="country" value="#{userData.selectedCountry}"
        size="20" />
</h:panelGrid>
</h:form>
</h:body>
</html>

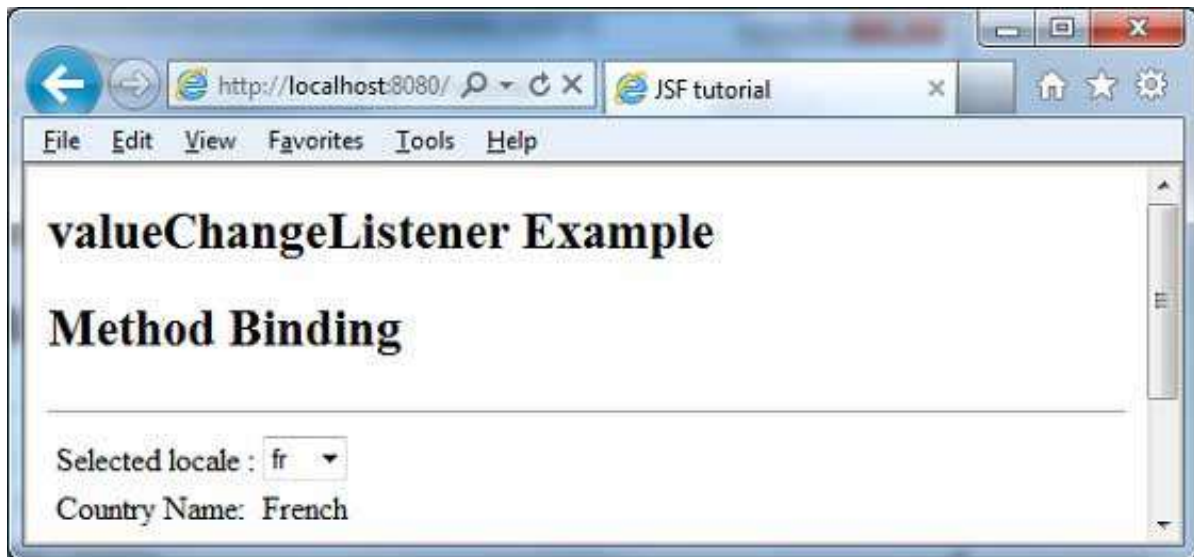
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.





Select locale. You will see the following result.



Modify **home.xhtml** again in the deployed directory where you've deployed the application as explained below. Keep the rest of the files unchanged.

#### home.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>valueChangeListener Examples</h2>
    <h:form>
      <h2>ValueChangeListener interface</h2>
      <hr/>
      <h:panelGrid columns="2">
        Selected locale :
        <h:selectOneMenu value="#{userData.selectedCountry}"
          onchange="submit()">
          <f:valueChangeListener
            type="com.tutorialspoint.test.LocaleChangeListener" />
          <f:selectItems value="#{userData.countries}" />
      </h:panelGrid>
    </h:form>
  </h:body>
</html>
```

```

        </h:selectOneMenu>
        Country Name:
        <h:outputText id="country1" value="#{userData.selectedCountry}"
            size="20" />
    </h:panelGrid>
</h:form>
</h:body>
</html>

```

Once you are ready with all the changes done, refresh the page in the browser. If everything is fine with your application, this will produce the following result.



Select locale. You will see the following result.



## actionListener

When the user interacts with the components, such as h:commandButton or h:link, the JSF fires action events which can be handled in two ways.

Technique	Description
<b>Method Binding</b>	Pass the name of the managed bean method in <i>actionListener</i> attribute of UI Component.
<b>ActionListener</b>	Implement ActionListener interface and pass the implementation class name to <i>actionListener</i> attribute of UI Component.

### Method Binding

#### Define a method

```
public void updateData(ActionEvent e){
    data="Hello World";
}
```

#### Use the above method

```
<h:commandButton id="submitButton"
    value="Submit" action="#{userData.showResult}"
    actionListener="#{userData.updateData}" />
</h:commandButton>
```

### ActionListener

#### Implement ActionListener

```
public class UserActionListener implements ActionListener{
    @Override
    public void processAction(ActionEvent arg0)
        throws AbortProcessingException {
        //access userData bean directly
        UserData userData = (UserData) FacesContext.getCurrentInstance().
            getExternalContext().getSessionMap().get("userData");
        userData.setData("Hello World");
    }
}
```

## Use listener method

```
<h:commandButton id="submitButton1"
    value="Submit" action="#{userData.showResult}" >
    <f:actionListener type="com.tutorialspoint.test.UserActionListener" />
</h:commandButton>
```

## Example Application

Let us create a test JSF application to test the actionListener in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>UserData.java</i> file as explained below.
3	Create <i>UserActionListener.java</i> file under a package <i>com.tutorialspoint.test</i> . Modify it as explained below.
4	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
5	Modify <i>result.xhtml</i> as explained below. Keep the rest of the files unchanged.
6	Compile and run the application to make sure the business logic is working as per the requirements.
7	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
8	Launch your web application using appropriate URL as explained below in the last step.

## UserData.java

```
package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.Map;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.event.ValueChangeEvent;
```

```

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private static Map<String,String> countryMap;
    private String data = "sample data";

    public String showResult(){
        return "result";
    }

    public void updateData(ActionEvent e){
        data="Hello World";
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}

```

### UserActionListener.java

```

package com.tutorialspoint.test;

import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;

public class UserActionListener implements ActionListener{
    @Override

```

```

public void processAction(ActionEvent arg0)
throws AbortProcessingException {
    //access userData bean directly
    UserData userData = (UserData) FacesContext.getCurrentInstance().
        getExternalContext().getSessionMap().get("userData");
    userData.setData("Hello World");
}
}

```

### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>actionListener Examples</h2>
        <h:form>
            <h2>Method Binding</h2>
            <hr/>
            <h:commandButton id="submitButton"
                value="Submit" action="#{userData.showResult}"
                actionListener="#{userData.updateData}" />
            </h:commandButton>
            <h2>ActionListener interface</h2>
            <hr/>
            <h:commandButton id="submitButton1"
                value="Submit" action="#{userData.showResult}" >
                <f:actionListener
                    type="com.tutorialspoint.test.UserActionListener" />
            </h:commandButton>
        </h:form>
    </h:body>
</html>

```

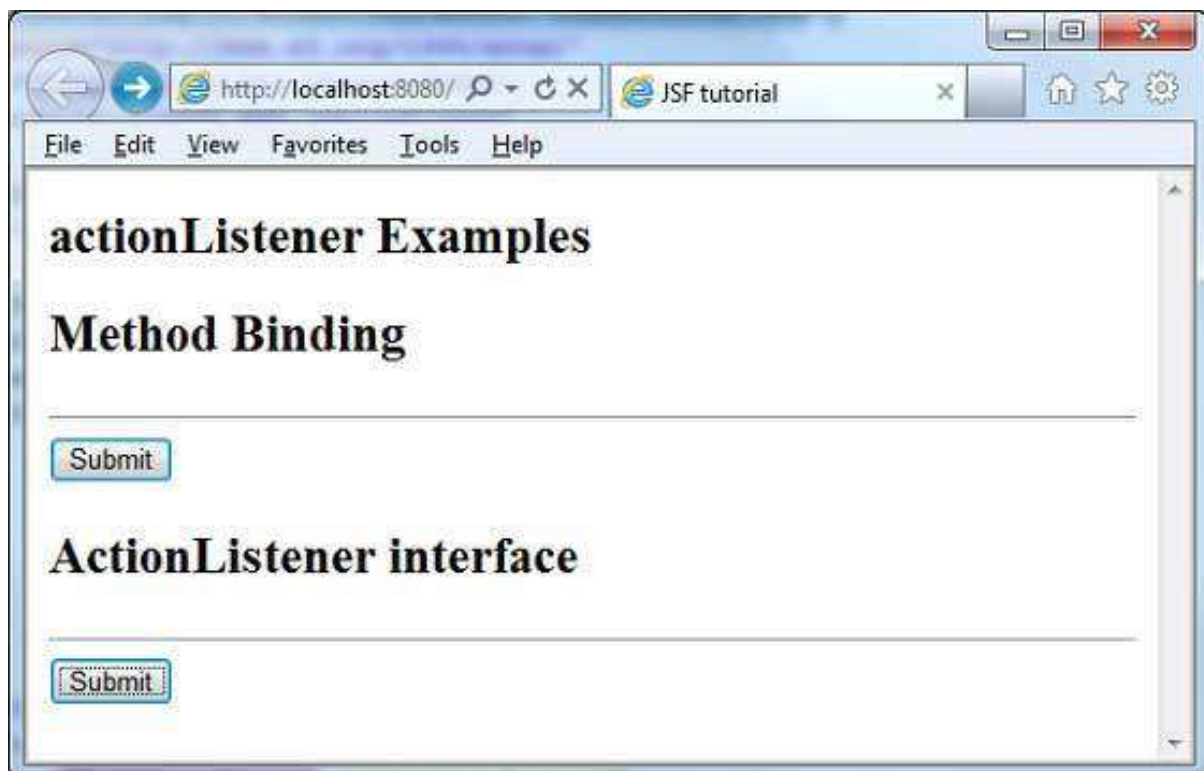
**result.xhtml**

```

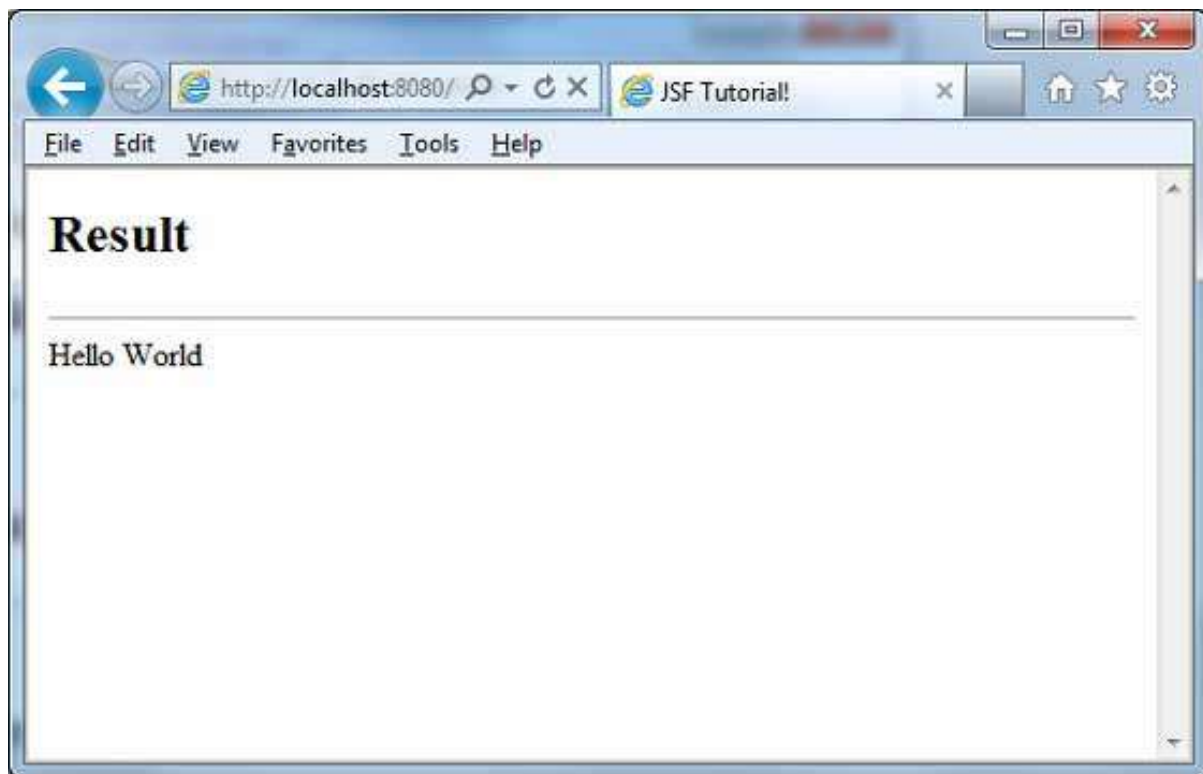
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>JSF Tutorial!</title>
  </h:head>
  <h:body>
    <h2>Result</h2>
    <hr />
    #{userData.data}
  </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Click any submit button. You will see the following result.



## Application Events

JSF provides system event listeners to perform application specific tasks during JSF application Life Cycle.

System Event	Description
PostConstructApplicationEvent	Fires when the application starts. Can be used to perform initialization tasks after the application has started.
PreDestroyApplicationEvent	Fires when the application is about to shut down. Can be used to perform cleanup tasks before the application is about to shut down.
PreRenderViewEvent	Fires before a JSF page is to be displayed. Can be used to authenticate the user and provide restricted access to JSF View.



System Events can be handled in the following manner.

Technique	Description
SystemEventListener	Implement SystemEventListener interface and register the system-event-listener class in faces-config.xml
Method Binding	Pass the name of the managed bean method in <i>listener</i> attribute of f:event.

## SystemEventListener

Implement SystemEventListener Interface.

```
public class CustomSystemEventListener implements SystemEventListener {
    @Override
    public void processEvent(SystemEvent event) throws
        AbortProcessingException {
        if(event instanceof PostConstructApplicationEvent){
            System.out.println("Application Started.
                PostConstructApplicationEvent occurred!");
        }
    }
}
```

Register custom system event listener for system event in faces-config.xml.

```
<system-event-listener>
    <system-event-listener-class>
        com.tutorialspoint.test.CustomSystemEventListener
    </system-event-listener-class>
    <system-event-class>

        javax.faces.event.PostConstructApplicationEvent
    </system-event-class>
</system-event-listener>
```

## Method Binding

Define a method.

```
public void handleEvent(ComponentSystemEvent event){
    data="Hello World";
}
```

Use the above method.

```
<f:event listener="#{user.handleEvent}" type="preRenderView" />
```

## Example Application

Let us create a test JSF application to test the system events in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>UserData.java</i> file as explained below.
3	Create <i>CustomSystemEventListener.java</i> file under a package <i>com.tutorialspoint.test</i> . Modify it as explained below
4	Modify <i>home.xhtml</i> as explained below.
5	Create <i>faces-config.xml</i> in <i>WEB-INF</i> folder. Modify it as explained below. Keep the rest of the files unchanged.
6	Compile and run the application to make sure the business logic is working as per the requirements.
7	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
8	Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private String data = "sample data";

    public void handleEvent(ComponentSystemEvent event){
        data="Hello World";
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```

**CustomSystemEventListener.java**

```
package com.tutorialspoint.test;

import javax.faces.application.Application;
import javax.faces.event.AbortProcessingException;

import javax.faces.event.PostConstructApplicationEvent;
```

```

import javax.faces.event.PreDestroyApplicationEvent;
import javax.faces.event.SystemEvent;
import javax.faces.event.SystemEventListener;

public class CustomSystemEventListener implements SystemEventListener {

    @Override
    public boolean isListenerForSource(Object value) {
        //only for Application
        return (value instanceof Application);
    }

    @Override
    public void processEvent(SystemEvent event)
        throws AbortProcessingException {
        if(event instanceof PostConstructApplicationEvent){
            System.out.println("Application Started.
            PostConstructApplicationEvent occurred!");
        }
        if(event instanceof PreDestroyApplicationEvent){
            System.out.println("PreDestroyApplicationEvent occurred.
            Application is stopping.");
        }
    }
}

```

### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>

    <h:body>

```

```

<h2>Application Events Examples</h2>
    <f:event listener="#{userData.handleEvent}" type="preRenderView" />
    #{userData.data}
</h:body>
</html>

```

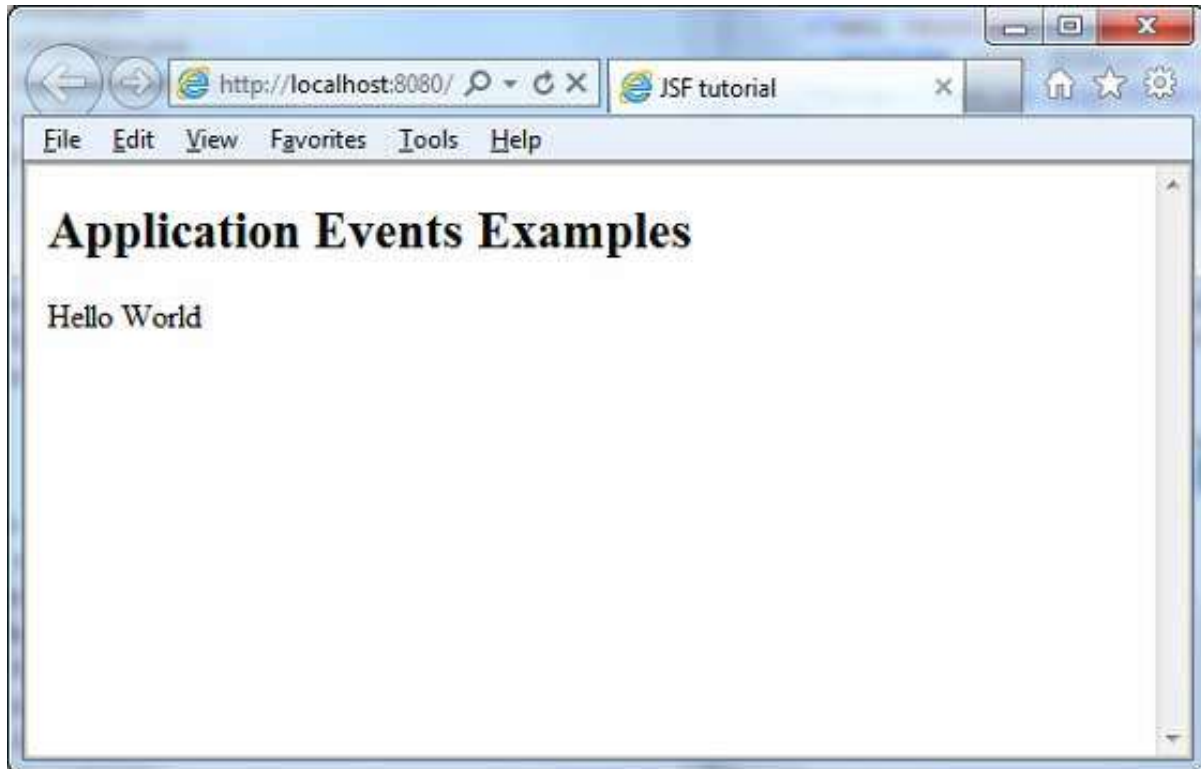
### faces-config.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<faces-config
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
    version="2.0">
    <application>
        <!-- Application Startup -->
        <system-event-listener>
            <system-event-listener-class>
                com.tutorialspoint.test.CustomSystemEventListener
            </system-event-listener-class>
            <system-event-class>
                javax.faces.event.PostConstructApplicationEvent
            </system-event-class>
        </system-event-listener>
        <!-- Before Application is to shut down -->
        <system-event-listener>
            <system-event-listener-class>
                com.tutorialspoint.test.CustomSystemEventListener
            </system-event-listener-class>
            <system-event-class>
                javax.faces.event.PreDestroyApplicationEvent
            </system-event-class>
        </system-event-listener>
    </application>
</faces-config>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Look into your web-server console output. You will see the following result.

```
INFO: Deploying web application archive helloworld.war
Dec 6, 2012 8:21:44 AM com.sun.faces.config.ConfigureListener
contextInitialized
INFO: Initializing Mojarra 2.1.7 (SNAPSHOT 20120206) for context '/helloworld'
Application Started. PostConstructApplicationEvent occurred!
Dec 6, 2012 8:21:46 AM com.sun.faces.config.ConfigureListener
$WebConfigResourceMonitor$Monitor <init>
INFO: Monitoring jndi:/localhost/helloworld/WEB-INF/faces-config.xml
for modifications
Dec 6, 2012 8:21:46 AM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Dec 6, 2012 8:21:46 AM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009

Dec 6, 2012 8:21:46 AM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/24 config=null
Dec 6, 2012 8:21:46 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 44272 ms
```

# 16. JSF - JDBC Integration

In this article, we'll demonstrate how to integrate database in JSF using JDBC.

Following are the database requirements to run this example.

Sr. No.	Software & Description
1	<b><u>PostgreSQL 9.1</u></b> Open Source and lightweight database
2	<b><u>PostgreSQL JDBC4 Driver</u></b> JDBC driver for PostgreSQL 9.1 and JDK 1.5 or above

Put PostgreSQL JDBC4 Driver jar in tomcat web server's lib directory.

## Database SQL Commands

```
create user user1;

create database testdb with owner=user1;

CREATE TABLE IF NOT EXISTS authors (
    id int PRIMARY KEY,
    name VARCHAR(25)
);

INSERT INTO authors(id, name) VALUES(1, 'Rob Bal');
INSERT INTO authors(id, name) VALUES(2, 'John Carter');
INSERT INTO authors(id, name) VALUES(3, 'Chris London');
INSERT INTO authors(id, name) VALUES(4, 'Truman De Bal');
INSERT INTO authors(id, name) VALUES(5, 'Emile Capote');
INSERT INTO authors(id, name) VALUES(7, 'Breech Jabber');
INSERT INTO authors(id, name) VALUES(8, 'Bob Carter');
INSERT INTO authors(id, name) VALUES(9, 'Nelson Mand');
INSERT INTO authors(id, name) VALUES(10, 'Tennant Mark');
```

```
alter user user1 with password 'user1';

grant all on authors to user1;
```

## Example Application

Let us create a test JSF application to test JDBC integration.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Create <i>resources</i> folder under <i>src -&gt; main</i> folder.
3	Create <i>css</i> folder under <i>src -&gt; main -&gt; resources</i> folder.
4	Create <i>styles.css</i> file under <i>src -&gt; main -&gt; resources -&gt; css</i> folder.
5	Modify <i>styles.css</i> file as explained below.
6	Modify <i>pom.xml</i> as explained below.
7	Create <i>Author.java</i> under package <i>com.tutorialspoint.test</i> as explained below.



8	Create <i>UserData.java</i> under package <i>com.tutorialspoint.test</i> as explained below.
9	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
10	Compile and run the application to make sure the business logic is working as per the requirements.
11	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
12	Launch your web application using appropriate URL as explained below in the last step.

### styles.css

```
.authorTable{
    border-collapse:collapse;
    border-bottom:1px solid #000000;
}

.authorTableHeader{
    text-align:center;
    background:none repeat scroll 0 0 #B5B5B5;
    border-bottom:1px solid #000000;
    border-top:1px solid #000000;
    padding:2px;
}

.authorTableOddRow{
    text-align:center;
    background:none repeat scroll 0 0 #FFFFFF;
}
```

```
.authorTableEvenRow{
    text-align:center;
    background:none repeat scroll 0 0 #D3D3D3;
}
```

### pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tutorialspoint.test</groupId>
  <artifactId>helloworld</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-api</artifactId>
      <version>2.1.7</version>
    </dependency>
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-impl</artifactId>
      <version>2.1.7</version>
    </dependency>
    <dependency>
```

```

        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
</dependencies>
<build>
    <finalName>helloworld</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.1</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-resources-plugin</artifactId>
            <version>2.6</version>
            <executions>
                <execution>
                    <id>copy-resources</id>
                    <phase>validate</phase>
                    <goals>
                        <goal>copy-resources</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>${basedir}/target/helloworld/resources
                    </outputDirectory>
                    <resources>
                        <resource>

```

```
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
    </resource>
</resources>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>
```

### Author.java

```
package com.tutorialspoint.test;

public class Author {
    int id;
    String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

**UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public List<Author> getAuthors(){
        ResultSet rs = null;
        PreparedStatement pst = null;
        Connection con = getConnection();
        String stm = "Select * from authors";
        List<Author> records = new ArrayList<Author>();
        try {
            pst = con.prepareStatement(stm);
            pst.execute();
            rs = pst.getResultSet();

            while(rs.next()){
                Author author = new Author();
                author.setId(rs.getInt(1));
                author.setName(rs.getString(2));
                records.add(author);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return records;
}

public Connection getConnection(){
    Connection con = null;

    String url = "jdbc:postgresql://localhost/testdb";
    String user = "user1";
    String password = "user1";
    try {
        con = DriverManager.getConnection(url, user, password);
        System.out.println("Connection completed.");
    } catch (SQLException ex) {
        System.out.println(ex.getMessage());
    }
    finally{
    }
    return con;
}
}

```

## home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>JSF Tutorial!</title>
        <h:outputStylesheet library="css" name="styles.css" />
    </h:head>
    <h2>JDBC Integration Example</h2>

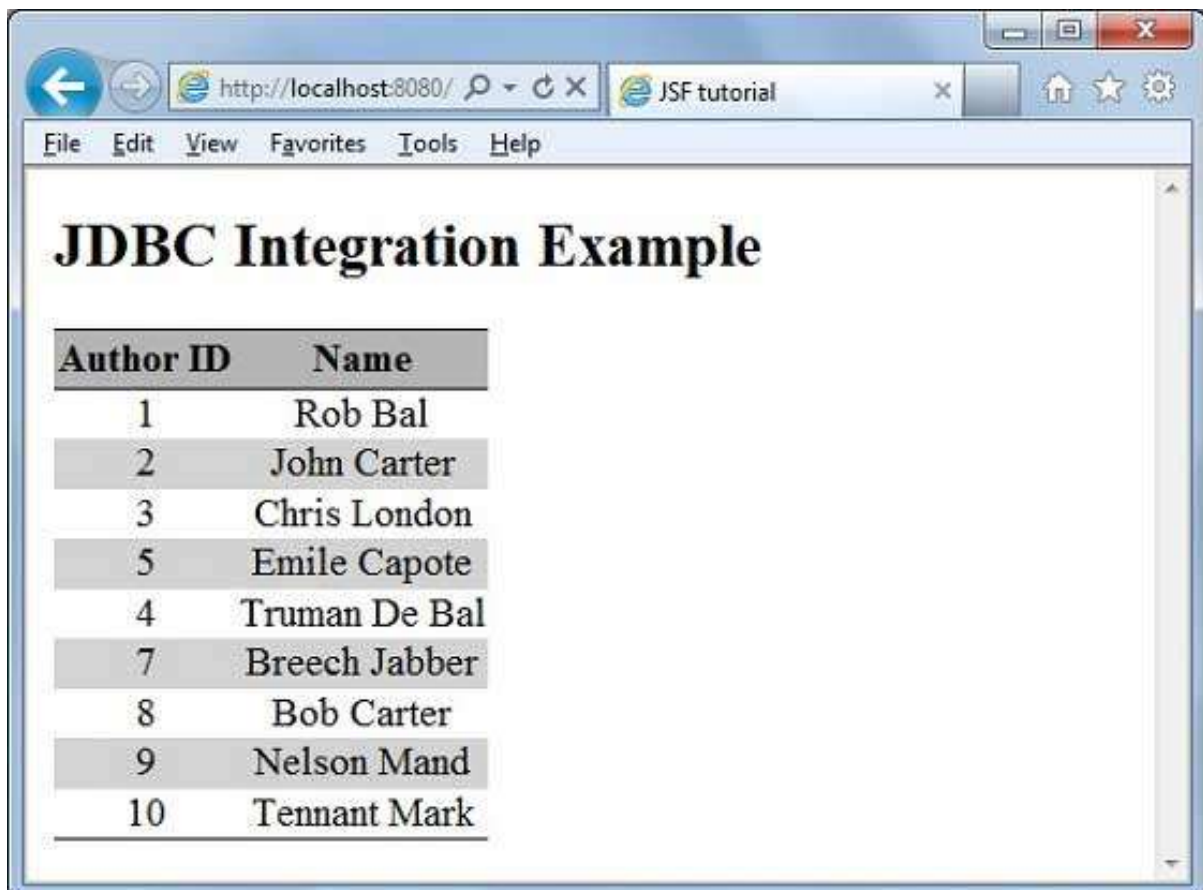
```

```

<h:dataTable value="#{userData.authors}" var="c"
  styleClass="authorTable"
  headerClass="authorTableHeader"
  rowClasses="authorTableOddRow,authorTableEvenRow">
  <h:column><f:facet name="header">Author ID</f:facet>
    #{c.id}
  </h:column>
  <h:column><f:facet name="header">Name</f:facet>
    #{c.name}
  </h:column>
</h:dataTable>
</h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



**JDBC Integration Example**

Author ID	Name
1	Rob Bal
2	John Carter
3	Chris London
5	Emile Capote
4	Truman De Bal
7	Breech Jabber
8	Bob Carter
9	Nelson Mand
10	Tennant Mark

# 17. JSF - Spring Integration

Spring provides special class `DelegatingVariableResolver` to integrate JSF and Spring together in a seamless manner.

Following steps are required to integrate Spring Dependency Injection (IOC) feature in JSF.

## Step 1: Add `DelegatingVariableResolver`

Add a `variable-resolver` entry in `faces-config.xml` to point to spring class **`DelegatingVariableResolver`**.

```
<faces-config>
  <application>
    <variable-resolver>
      org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
    ...
  </faces-config>
```

## Step 2: Add Context Listeners

Add **`ContextLoaderListener`** and **`RequestContextListener`** listener provided by spring framework in `web.xml`.

```
<web-app>
  ...
  <!-- Add Support for Spring -->
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <listener>
    <listener-class>
      org.springframework.web.context.request.RequestContextListener
    </listener-class>
  </listener>
  ...
</web-app>
```



### Step 3: Define Dependency

Define bean(s) in applicationContext.xml which will be used as dependency in managed bean.

```
<beans>
  <bean id="messageService"
    class="com.tutorialspoint.test.MessageServiceImpl">
    <property name="message" value="Hello World!" />
  </bean>
</beans>
```

### Step 4: Add Dependency

**DelegatingVariableResolver** first delegates value lookups to the default resolver of the JSF and then to Spring's WebApplicationContext. This allows one to easily inject spring-based dependencies into one's JSF-managed beans.

We've injected messageService as spring-based dependency here.

```
<faces-config>
  ...
  <managed-bean>
    <managed-bean-name>userData</managed-bean-name>
    <managed-bean-class>com.tutorialspoint.test.UserData</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
      <property-name>messageService</property-name>
      <value>#{messageService}</value>
    </managed-property>
  </managed-bean>
</faces-config>
```

### Step 5: Use Dependency

```
//jsf managed bean
public class UserData {
  //spring managed dependency
  private MessageService messageService;

  public void setMessageService(MessageService messageService) {
    this.messageService = messageService;
  }
}
```

```

    }

    public String getGreetingMessage(){
        return messageService.getGreetingMessage();
    }
}

```

## Example Application

Let us create a test JSF application to test spring integration.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>pom.xml</i> as explained below.
3	Create <i>faces-config.xml</i> in <i>WEB-INF</i> folder as explained below.
4	Modify <i>web.xml</i> as explained below.
5	Create <i>applicationContext.xml</i> in <i>WEB-INF</i> folder as explained below.
6	Create <i>MessageService.java</i> under package <i>com.tutorialspoint.test</i> as explained below.

7	Create <i>MessageServiceImpl.java</i> under package <i>com.tutorialspoint.test</i> as explained below.
8	Create <i>UserData.java</i> under package <i>com.tutorialspoint.test</i> as explained below.
9	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
10	Compile and run the application to make sure the business logic is working as per the requirements.
11	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
12	Launch your web application using appropriate URL as explained below in the last step.

### pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tutorialspoint.test</groupId>
  <artifactId>helloworld</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
```

```

        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-api</artifactId>
        <version>2.1.7</version>
    </dependency>
    <dependency>
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-impl</artifactId>
        <version>2.1.7</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>3.1.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>3.1.2.RELEASE</version>
    </dependency>
</dependencies>
<build>
    <finalName>helloworld</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.1</version>

```

```

        <configuration>
            <source>1.6</source>
            <target>1.6</target>
        </configuration>
    </plugin>
    <plugin>
        <artifactId>maven-resources-plugin</artifactId>
        <version>2.6</version>
        <executions>
            <execution>
                <id>copy-resources</id>
                <phase>validate</phase>
                <goals>
                    <goal>copy-resources</goal>
                </goals>
                <configuration>
                    <outputDirectory>${basedir}/target/helloworld/resources
                    </outputDirectory>
                    <resources>
                        <resource>
                            <directory>src/main/resources</directory>
                            <filtering>true</filtering>
                        </resource>
                    </resources>
                </configuration>
            </execution>
        </executions>
    </plugin>
</plugins>
</build>
</project>

```

**faces-config.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<faces-config
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
  version="2.0">
  <application>
    <variable-resolver>
      org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
  </application>
  <managed-bean>
    <managed-bean-name>userData</managed-bean-name>
    <managed-bean-class>com.tutorialspoint.test.UserData</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
      <property-name>messageService</property-name>
      <value>#{messageService}</value>
    </managed-property>
  </managed-bean>
</faces-config>

```

**web.xml**

```

<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>
  <display-name>Archetype Created Web Application</display-name>

  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <!-- Add Support for Spring -->

```

```

<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
<listener>
    <listener-class>
        org.springframework.web.context.request.RequestContextListener
    </listener-class>
</listener>
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
</web-app>

```

### applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
    "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="messageService"
        class="com.tutorialspoint.test.MessageServiceImpl">
        <property name="message" value="Hello World!" />
    </bean>
</beans>

```

### MessageService.java

```

package com.tutorialspoint.test;

public interface MessageService {
    String getGreetingMessage();
}

```

### MessageServiceImpl.java

```
package com.tutorialspoint.test;

public class MessageServiceImpl implements MessageService {

    private String message;

    public String getGreetingMessage() {
        return message;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

### **UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;

public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private MessageService messageService;

    public MessageService getMessageService() {
        return messageService;
    }

    public void setMessageService(MessageService messageService) {
        this.messageService = messageService;
    }
}
```



```

    public String getGreetingMessage(){
        return messageService.getGreetingMessage();
    }
}

```

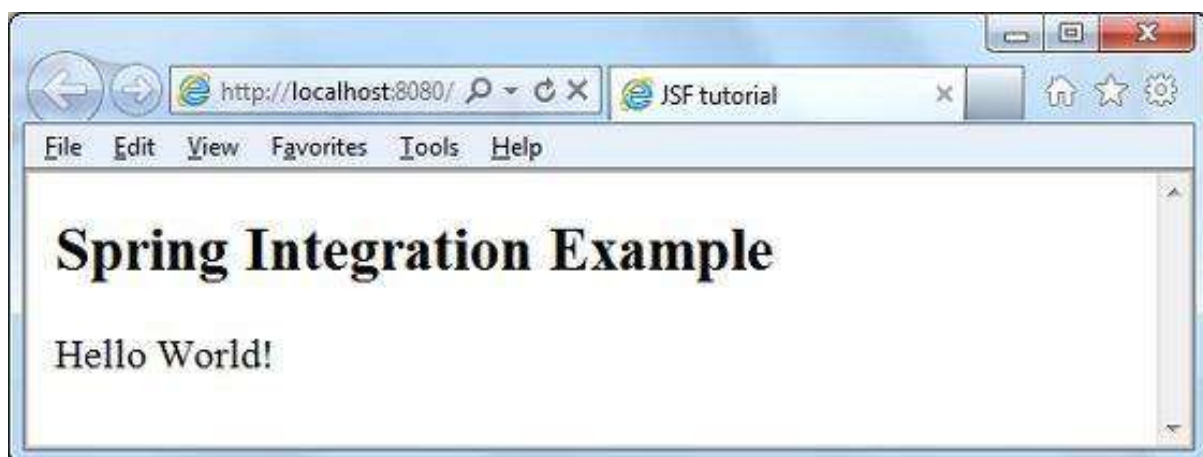
### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>JSF Tutorial!</title>
  </h:head>
  <h:body>
    <h2>Spring Integration Example</h2>
    #{userData.greetingMessage}
  </h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



# 18. JSF - Expression Language

JSF provides a rich expression language. We can write normal operations using **`#{operation-expression}`** notation. Following are some of the advantages of JSF Expression languages.

- Can reference bean properties where bean can be an object stored in request, session or application scope or is a managed bean.
- Provides easy access to elements of a collection which can be a list, map or an array.
- Provides easy access to predefined objects such as a request.
- Arithmetic, logical and relational operations can be done using expression language.
- Automatic type conversion.
- Shows missing values as empty strings instead of `NullPointerException`.

## Example Application

Let us create a test JSF application to test expression language.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Modify <i>UserData.java</i> under package <i>com.tutorialspoint.test</i> as explained below.
3	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
4	Compile and run the application to make sure the business logic is working as per the requirements.

5	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
6	Launch your web application using appropriate URL as explained below in the last step.

### **UserData.java**

```
package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.Date;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private Date createTime = new Date();
    private String message = "Hello World!";

    public Date getCreateTime() {
        return(createTime);
    }

    public String getMessage() {
        return(message);
    }

}
```

**home.xhtml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>JSF Tutorial!</title>
  </h:head>
  <h2>Expression Language Example</h2>
  Creation time:
  <h:outputText value="#{userData.createTime}"/>
  <br/><br/>Message:
  <h:outputText value="#{userData.message}"/>
  </h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



# 19. JSF - Internationalization

Internationalization is a technique in which status messages, GUI component labels, currency, date are not hardcoded in the program. Instead, they are stored outside the source code in resource bundles and retrieved dynamically. JSF provides a very convenient way to handle resource bundle.

Following steps are required to internalize a JSF application.

## Step 1: Define properties files

Create properties file for each locale. Name should be in <file-name>\_<locale>.properties format.

Default locale can be omitted in file name.

### messages.properties

```
greeting=Hello World!
```

### messages\_fr.properties

```
greeting=Bonjour tout le monde!
```

## Step 2: Update faces-config.xml

### faces-config.xml

```
<application>
  <locale-config>
    <default-locale>en</default-locale>
    <supported-locale>fr</supported-locale>
  </locale-config>
  <resource-bundle>
    <base-name>com.tutorialspoint.messages</base-name>
    <var>msg</var>
  </resource-bundle>
</application>
```

### Step 3: Use resource-bundle var

home.xhtml

```
<h:outputText value="#{msg['greeting']}" />
```

### Example Application

Let us create a test JSF application to test internationalization in JSF.

Step	Description
1	Create a project with a name <i>helloworld</i> under a package <i>com.tutorialspoint.test</i> as explained in the <i>JSF - First Application</i> chapter.
2	Create <i>resources</i> folder under <i>src -&gt; main</i> folder.
3	Create <i>com</i> folder under <i>src -&gt; main -&gt; resources</i> folder.
4	Create <i>tutorialspoint</i> folder under <i>src -&gt; main -&gt; resources -&gt; com</i> folder.
5	Create <i>messages.properties</i> file under <i>src -&gt; main -&gt; resources -&gt; com -&gt; tutorialspoint</i> folder. Modify it as explained below.
6	Create <i>messages_fr.properties</i> file under <i>src -&gt; main -&gt; resources -&gt; com -&gt; tutorialspoint</i> folder. Modify it as explained below.
7	Create <i>faces-config.xml</i> in <i>WEB-INF</i> folder as explained below.
8	Create <i>UserData.java</i> under package <i>com.tutorialspoint.test</i> as explained below.

9	Modify <i>home.xhtml</i> as explained below. Keep the rest of the files unchanged.
10	Compile and run the application to make sure the business logic is working as per the requirements.
11	Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.
12	Launch your web application using appropriate URL as explained below in the last step.

### messages.properties

```
greeting=Hello World!
```

### messages\_fr.properties

```
greeting=Bonjour tout le monde!
```

### faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
  version="2.0">
  <application>
    <locale-config>
      <default-locale>en</default-locale>
      <supported-locale>fr</supported-locale>
    </locale-config>
    <resource-bundle>
```

```

        <base-name>com.tutorialspoint.messages</base-name>
        <var>msg</var>
    </resource-bundle>
</application>
</faces-config>

```

## UserData.java

```

package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.Locale;
import java.util.Map;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.faces.event.ValueChangeEvent;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private String locale;

    private static Map<String, Object> countries;

    static{
        countries = new LinkedHashMap<String, Object>();
        countries.put("English", Locale.ENGLISH);
        countries.put("French", Locale.FRENCH);
    }

    public Map<String, Object> getCountries() {
        return countries;
    }
}

```



```

public String getLocale() {
    return locale;
}

public void setLocale(String locale) {
    this.locale = locale;
}

//value change event listener
public void localeChanged(ValueChangeEvent e){
    String newLocaleValue = e.getNewValue().toString();
    for (Map.Entry<String, Object> entry : countries.entrySet()) {
        if(entry.getValue().toString().equals(newLocaleValue)){
            FacesContext.getCurrentInstance()
                .getViewRoot().setLocale((Locale)entry.getValue());
        }
    }
}
}
}

```

### home.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
>
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>Internalization Language Example</h2>
        <h:form>
            <h3><h:outputText value="#{msg['greeting']}" /></h3>

            <h:panelGrid columns="2">

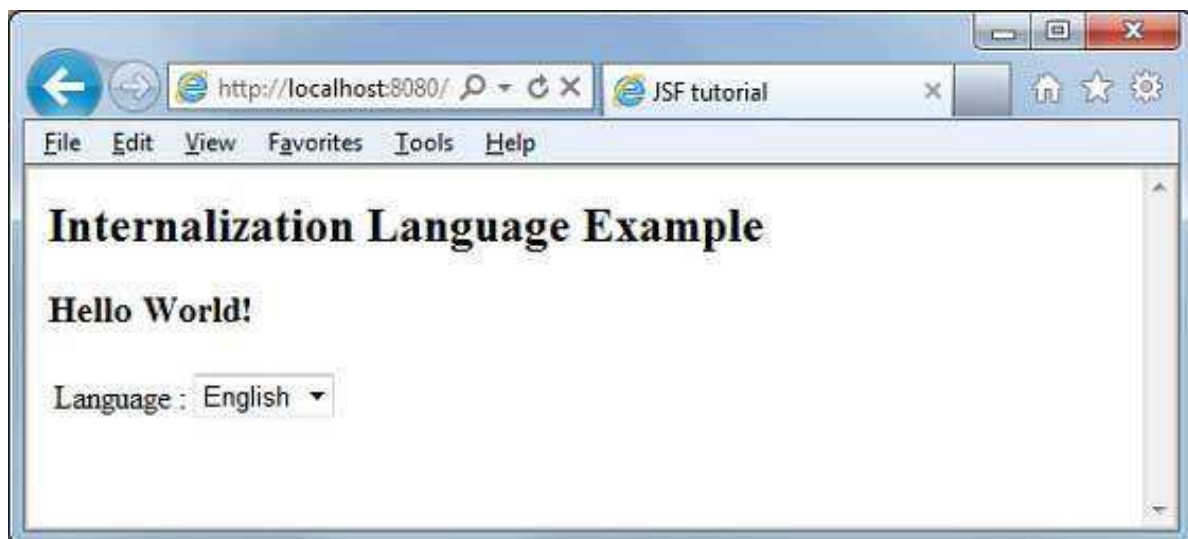
```

```

        Language :
        <h:selectOneMenu value="#{userData.locale}" onchange="submit()"
            valueChangeListener="#{userData.localeChanged}">
            <f:selectItems value="#{userData.countries}" />
        </h:selectOneMenu>
    </h:panelGrid>
</h:form>
</h:body>
</html>

```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.



Change language from dropdown. You will see the following output.

