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.
# 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** .................................................................................................................................................. 25
   - @ManagedBean Annotation .............................................................................................................................................. 26
   - Scope Annotations .............................................................................................................................................................. 26
   - @ManagedProperty Annotation ....................................................................................................................................... 26

7. **JSF – PAGE NAVIGATION** ............................................................................................................................................... 30
   - Implicit Navigation ............................................................................................................................................................ 30
   - Auto Navigation in JSF Page ............................................................................................................................................ 30
   - Auto Navigation in Managed Bean .................................................................................................................................... 31
   - Conditional Navigation ...................................................................................................................................................... 32
   - Resolving Navigation Based on from-action ...................................................................................................................... 34
   - Forward vs Redirect ......................................................................................................................................................... 36

8. **JSF – BASIC TAGS** ............................................................................................................................................................. 46
   - h:inputText ......................................................................................................................................................................... 48
   - h:inputSecret ................................................................................................................................................................. 55
   - h:inputTextarea ............................................................................................................................................................ 60
9. JSF – FACELET TAGS ................................................................................................................. 205

Template Tags ................................................................................................................................. 206

Creating Template .......................................................................................................................... 207

ui:param Tag ................................................................................................................................. 214

Parameter to Section of a Template ............................................................................................. 215
Parameter to Template .............................................................................................................. 215
Custom Tag ............................................................................................................................... 219
ui:remove Tag ............................................................................................................................ 226

10. JSF - CONVERTOR TAGS ............................................................................................... 229
    f:convertNumber ............................................................................................................... 229
    f:convertDateTime ............................................................................................................. 233
    Custom Converter ............................................................................................................... 238

11. JSF - VALIDATOR TAGS .................................................................................................. 247
    f:validateLength ............................................................................................................... 248
    f:validateLongRange ......................................................................................................... 252
    f:validateDoubleRange ..................................................................................................... 255
    f:validateRegex ................................................................................................................ 259
    Custom Validator ............................................................................................................... 264

12. JSF – DATATABLE ............................................................................................................. 273
    Display DataTable .............................................................................................................. 274
    Add Data to DataTable ....................................................................................................... 285
    Edit Data of a DataTable ................................................................................................... 290
    Delete Data of a DataTable ............................................................................................... 294
    Using DataModel in a DataTable ..................................................................................... 297

13. JSF – COMPOSITE COMPONENTS ................................................................................ 301
    Define Custom Component .............................................................................................. 301
    Use Custom Component.................................................................................................... 303

14. JSF – AJAX ..................................................................................................................... 309
15. JSF – EVENT HANDLING ......................................................................................................... 315

valueChangeListener ...................................................................................................................... 316

actionListener ................................................................................................................................ 324

Application Events .......................................................................................................................... 331

16. JSF - JDBC INTEGRATION........................................................................................................ 339

17. JSF - SPRING INTEGRATION .................................................................................................. 349

18. JSF - EXPRESSION LANGUAGE ............................................................................................ 360

19. JSF - INTERNATIONALIZATION ............................................................................................ 364
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
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.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDK</td>
<td>1.5 or above</td>
</tr>
<tr>
<td>Memory</td>
<td>No minimum requirement</td>
</tr>
<tr>
<td>Disk Space</td>
<td>No minimum requirement</td>
</tr>
<tr>
<td>Operating System</td>
<td>No minimum requirement</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>OS</th>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Open Command Console</td>
<td>c:&gt; java -version</td>
</tr>
<tr>
<td>Linux</td>
<td>Open Command Terminal</td>
<td>$ java -version</td>
</tr>
<tr>
<td>Mac</td>
<td>Open Terminal</td>
<td>machine:~ joseph$ java -version</td>
</tr>
</tbody>
</table>
Let's verify the output for all the operating systems:

<table>
<thead>
<tr>
<th>OS</th>
<th>Generated Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>java version &quot;1.6.0_21&quot;</td>
</tr>
<tr>
<td></td>
<td>Java(TM) SE Runtime Environment (build 1.6.0_21-b07)</td>
</tr>
<tr>
<td></td>
<td>Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)</td>
</tr>
<tr>
<td>Linux</td>
<td>java version &quot;1.6.0_21&quot;</td>
</tr>
<tr>
<td></td>
<td>Java(TM) SE Runtime Environment (build 1.6.0_21-b07)</td>
</tr>
<tr>
<td></td>
<td>Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)</td>
</tr>
<tr>
<td>Mac</td>
<td>java version &quot;1.6.0_21&quot;</td>
</tr>
<tr>
<td></td>
<td>Java(TM) SE Runtime Environment (build 1.6.0_21-b07)</td>
</tr>
<tr>
<td></td>
<td>Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)</td>
</tr>
</tbody>
</table>

**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](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html). 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 -

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
</table>


Append Java compiler location to System Path.

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

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/](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/](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](http://maven.apache.org/download.html)

<table>
<thead>
<tr>
<th>OS</th>
<th>Archive name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>apache-maven-2.0.11-bin.zip</td>
</tr>
<tr>
<td>Linux</td>
<td>apache-maven-2.0.11-bin.tar.gz</td>
</tr>
<tr>
<td>Mac</td>
<td>apache-maven-2.0.11-bin.tar.gz</td>
</tr>
</tbody>
</table>

### 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.

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

### Step 6: Set Maven environment variables.

Add M2_HOME, M2, MAVEN_OPTS to environment variables.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set the environment variables using system properties.</td>
</tr>
</tbody>
</table>
Step 7: Add Maven bin directory location to system path.
Now append M2 variable to System Path.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Append the string ;%M2% to the end of the system variable, Path.</td>
</tr>
<tr>
<td>Linux</td>
<td>export PATH=$M2:$PATH</td>
</tr>
<tr>
<td>Mac</td>
<td>export PATH=$M2:$PATH</td>
</tr>
</tbody>
</table>

Step 8: Verify Maven installation.
Open console, execute the following `mvn` command.
Step 9: Set Up Apache Tomcat.

You can download the latest version of Tomcat from [http://tomcat.apache.org/](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
Tomcat can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine.

```bash
%CATALINA_HOME%/bin/startup.bat
```

or

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

After a successful startup, the default web applications included with Tomcat will be available by visiting [http://localhost:8080/](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
```
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:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Carries Data and login</td>
</tr>
<tr>
<td>View</td>
<td>Shows User Interface</td>
</tr>
<tr>
<td>Controller</td>
<td>Handles processing of an application</td>
</tr>
</tbody>
</table>

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 up to 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.
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] ------------------------------------------------------------------------
[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
```

...
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.

<table>
<thead>
<tr>
<th>Folder Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>helloworld</td>
<td>Contains src folder and pom.xml</td>
</tr>
<tr>
<td>src/main/wepapp</td>
<td>Contains WEB-INF folder and index.jsp page</td>
</tr>
<tr>
<td>src/main/resources</td>
<td>It contains images/properties files (In the above example, we need to create</td>
</tr>
<tr>
<td></td>
<td>this structure manually)</td>
</tr>
</tbody>
</table>
Add JSF Capability to Project

Add the following JSF dependencies.

```xml
<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

```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>
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
launching.JRE_CONTAINER
Downloading: http://repo.maven.apache.org/
```
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
<?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>
</web-app>
```
<servlet>
    <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.

```java
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.

```xml
<!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] Building helloworld Maven Webapp
[INFO] [INFO] Id: com.tutorialspoint.test:helloworld:war:1.0-SNAPSHOT
[INFO] [INFO] task-segment: [package]
[INFO] [INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [INFO] [compiler:testCompile]
[INFO] No sources to compile
[INFO] [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.
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

```xml
<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

```java
@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.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@RequestScoped</td>
<td>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.</td>
</tr>
<tr>
<td>@NoneScoped</td>
<td>Bean lives as long as a single EL evaluation. It gets created upon an EL evaluation and gets destroyed immediately after the EL evaluation.</td>
</tr>
<tr>
<td>@ViewScoped</td>
<td>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.</td>
</tr>
<tr>
<td>@SessionScoped</td>
<td>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.</td>
</tr>
<tr>
<td>@ApplicationScoped</td>
<td>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 eager=true attribute is set in @ManagedBean) and gets destroyed when the web application shuts down.</td>
</tr>
<tr>
<td>@CustomScoped</td>
<td>Bean lives as long as the bean's entry in the custom Map, which is created for this scope lives.</td>
</tr>
</tbody>
</table>
@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.

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

HelloWorld.java

```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() {
```
```java
    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**

```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**

```xml
<!DOCTYPE html PUBLIC "-/W3C//DTD XHTML 1.0 Transitional//EN"
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.
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.

```java
@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.

```html
<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.

```java
@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.

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

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

```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.
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.

![Image of Page1](image)

**Example Application**

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

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

```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>
<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:form>
  </h:body>
</html>
<h:commandButton action="page2" value="Page2" />

</h:form>
<br/>

<h2>Conditional Navigation</h2>
<br/>

<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>
<br/>

<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>
<br/>
<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>
End of ebook preview
If you liked what you saw...
Buy it from our store @ https://store.tutorialspoint.com