

# JSF Primefaces



 JournalDev Pankaj Kumar

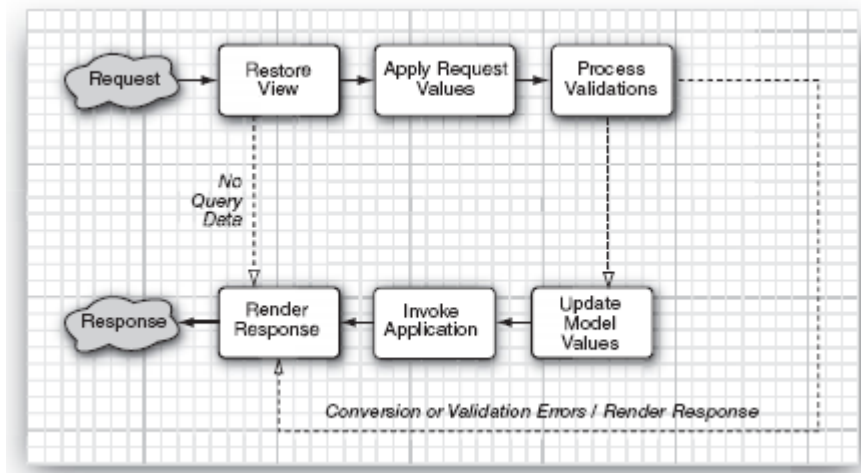
# Table of Content

1. JSF 2 & Primefaces 5.....	2
A. What's New in JSF 2 .....	3
a. Used Tools for Completing Tutorial .....	3
b. Creating Eclipse Project.....	4
c. JSF Installation & Configuration .....	5
B. Primefaces 5 Installation .....	7
a. Developing Primefaces Application .....	7
b. Convert into Maven .....	11
2. Primefaces AccordionPanel Component Tutorial .....	14
A. AccordionPanel Basic Info .....	14
B. AccordionPanel Attributes .....	15
C. Create Project Using Eclipse IDE .....	16
D. Setting up Required Files & Dependencies .....	18
E. Getting Started With AccordionPanel .....	22
F. AccordionPanel – Dynamic Content Loading .....	24
G. AccordionPanel – Ajax Behavior Events.....	27
H. AccordionPanel – Client Side API.....	29
3. Primefaces Spring & Hibernate Integration.....	33
A. Required Tools .....	33
B. Primefaces Spring Hibernate Project Structure.....	34
a. Create Database Employee Table .....	34
b. Create Employee Domain Class .....	35
C. Setting Up Maven.....	36
D. Setting Up Hibernate.....	38
E. Test an Application.....	40
F. Setting Up Spring .....	42
G. Spring EmployeeService.....	47
H. Primefaces Managed Bean – RegisterEmployee .....	49
4. Primefaces, Spring 4 with JPA (Hibernate 4/EclipseLink) Example .....	52
A. Final Project Structure .....	53
a. Database Tables .....	53
b. Domain Classes.....	54
2. Persistence Unit .....	56
3. Maven Dependencies .....	57
4. Hibernate/JPA Spring Configuration.....	59
5. EclipseLink/JPA Spring Configuration .....	61
6. Primefaces Deployment Descriptor .....	61
7. Spring EmployeeService.....	62
8. Primefaces Managed Bean – RegisterEmployee .....	63
9. Primefaces Employee Registration .....	64
Copyright Notice.....	67
References.....	68

# 1. JSF 2 & Primefaces 5

[JavaServer Faces](#) is one of the leading framework that is used these days for implementing Java web application user interface. JSF has componentized web application and especially that part related to the interface, in that all single view in the jsf has been built using a server side tree of components decoded into HTML when it comes to be rendered into browser.

The process of rendering the view in JSF does pass through what known as JSF lifecycle. This tutorial isn't intended for providing you a detailed discussion of how lifecycle works or how could we deal with. It's just a notification about what you should know about the JSF framework and how get JSF view ready for rendering.



JsF has two major implementations till the time in which the article written, oracle implementation **Mojarra** and Apache **MyFaces** implementation. Several JsF libraries has been coming into existence, Richfaces, IceFaces, [Primefaces](#), MyFaces, etc and one of the most lead library that used intensively and has an excellent reputation is **Primefaces**. Primefaces cellabrate before months ago by releasing the Primefaces 5 which will consider the subject of this tutorial and next coming tutorials.

For being able of using the primefaces 5, you must install and configure it into your project. Either you are going to use a simple text editor or an enterprise

development environment, by ending of this tutorial you will be ready for discovering the all Primefaces components.

## A. What's New in JSF 2

As we knew, a JavaServer Faces is a framework for developing rich user interface web pages. JSF has been introduced in several Java Community Request JSR where the final release of [JSF 2](#) was released in Jul, 2009 which contains a set of enhancement and new functionalities. Set of consequences have followed JSF 2 and the final one was [JSF 2.2](#) that released in May 2013.

Unlike JSF 1.x, JSF 2.x has been coming with a lot of features like using the annotations for declaring the JSF managed beans, Converters, Validators, Scopes etc. That's not all the story, JSF 2 had provided a newly scopes like View Scope, Custom Scope, Flow Scope and Conversation Scope and much more.

Also, we cannot forget the most amazing feature that added for JSF 2 and it's an Ajax concept. In JSF 2, Ajax has built into JSF framework inherently. So, any of JSF component can be ajaxified by simply adding the Ajax stuff. Navigation rules also has changed and be much easier as well.

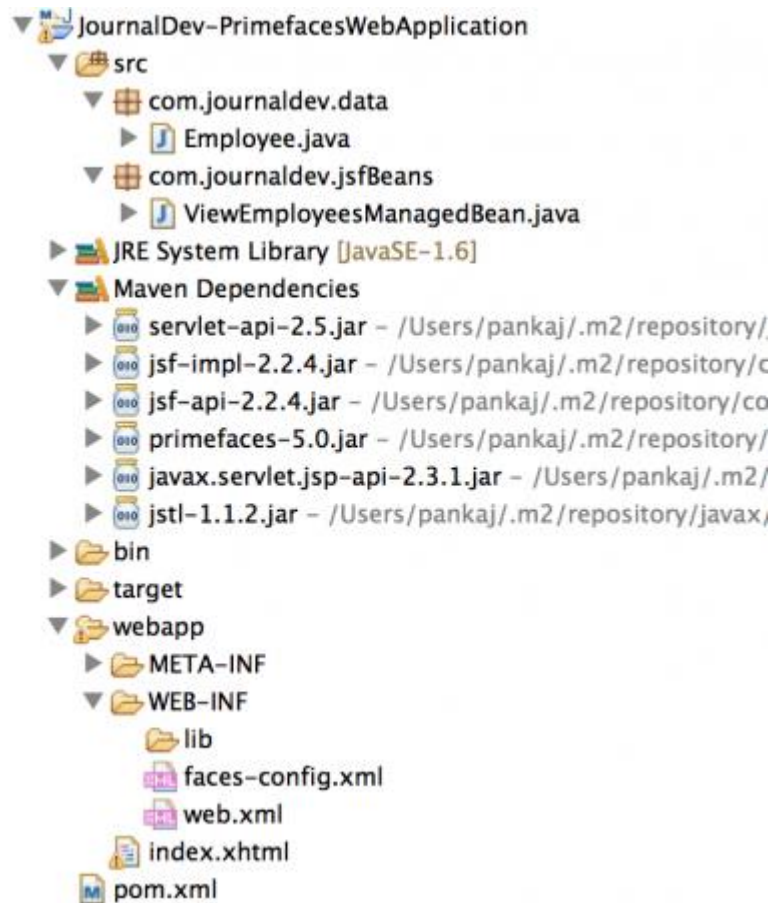
Next coming tutorials would cover more about those features that added for JSF 2, while in this tutorial, you're going to create simple JSF application and a bsic sample of how we could use the Primefaces Tags for implementing certain business scenario.

### a. Used Tools for Completing Tutorial

For getting started discovering this tutorial, you have to use the following development tools.

1. Tomcat 7
2. Eclipse IDE
3. Maven 3
4. JSF 2 / Primefaces 5

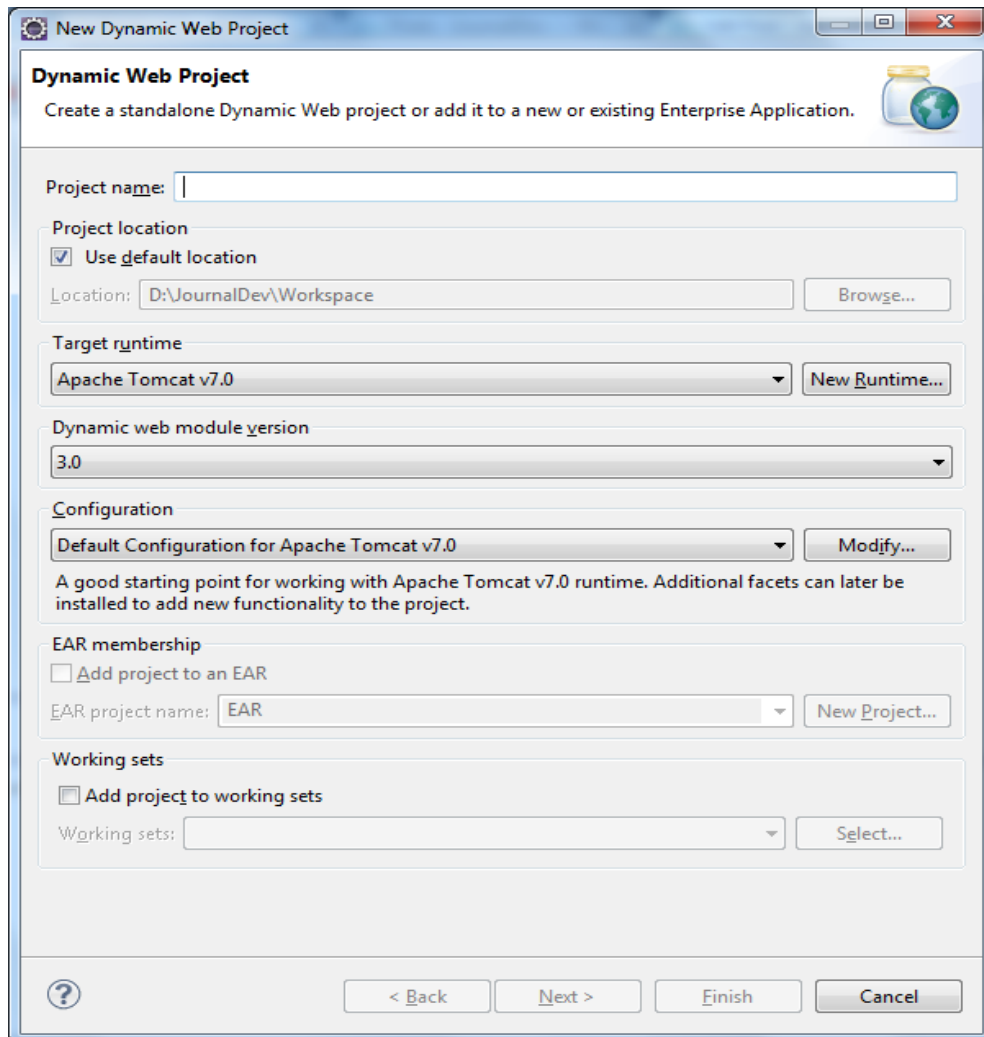
It's obvious that we've used the Tomcat 7 for deploying the application and Eclipse IDE for developing the required components where the Maven used as building tool and for managing dependencies. So, be sure that you are aware of how could be all of these software installed and configured into your development machine. Our final project will look like below image.



## b. Creating Eclipse Project

Eclipse IDE support the development of web project under **Dynamic Project** umbrella. For creating a dynamic project just follow the below steps:

- Open Eclipse IDE
- Right-Click on the project explorer space and select **New – Dynamic Web Project**



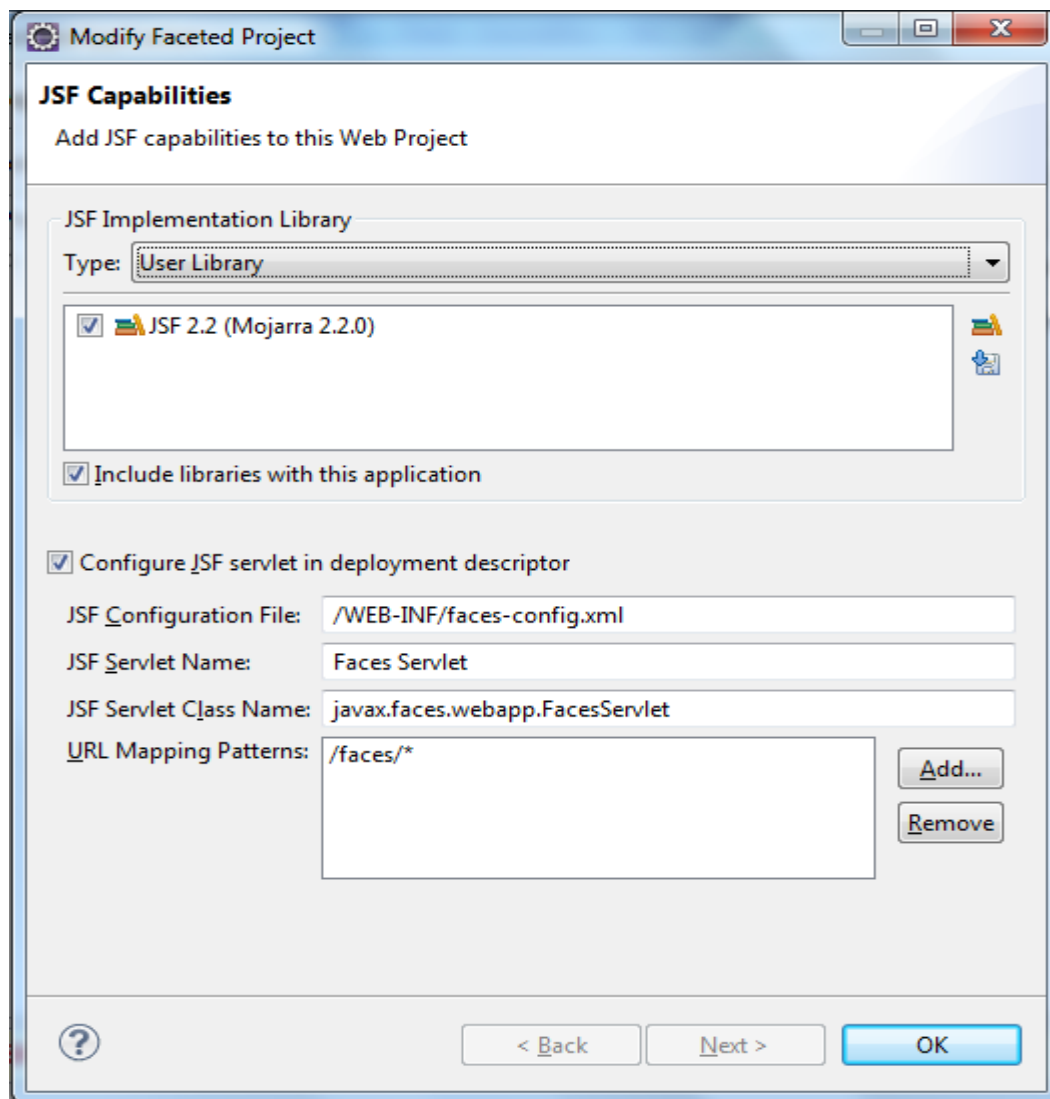
- Complete the project creation process by setting up the project name, target runtime, source folder, context root, content directory and web.xml

## c. JSF Installation & Configuration

As we've mentioned earlier, our goal is to use JSF/Primefaces for developing web application that uses the primefaces user interface component, but for now, all what we had is just a simple dynamic application that needs more steps for being jsf configured. To add a jsf into your project you need to add the **jsf facet** and making notice that the adding of jsf implementation does help you build a jsf application that uses Mojarra. For adding that facet you need to follow the below steps:

1. Open the properties window for the created project
2. From the Project Facets pane, just make the JavaServer Faces checked and follow **Further configuration required** for completing the configuration
3. Once you've clicked on **Further configuration required**, JSF Capabilities window must be shown
4. Adding the jsf implementation library by clicking on **Download Library** and select from the opening window JSF 2.2 (Mojarra 2.2.0)

After installing the jsf library, the jsf capabilities window looks like



By end of this phase, you have a web application with jsf capabilities.

## B. Primefaces 5 Installation

For now, your application is ready for using a JavaServer Faces User Interface, but not using of Primefaces. For being able of using the primefaces, you have to follow the below steps:

1. Download the required primefaces library from the [primefaces](#) official site

Or from Maven central.

2. Include the downloaded Primefaces JAR into your lib folder that beneath of WEB-INF folder

### a. Developing Primefaces Application

Now, your project is ready for developing a JSF/Primefaces application as you would see. We are going to create a simple application in which a Primefaces **DataTable** has consumed a list of Employees from the backing bean. Employees list would be populated by a **@PostConstruct** special method. Follow the below steps for developing a full JSF/Primefaces application.

1. Create a managed bean named **ViewEmployeesManagedBean**
2. Create a Pojo named Employee that contains **EmployeeName** and **EmployeeId**
3. Create a Primefaces view in order to consume the employees list in the defined managed bean

```
package com.journaldev.data;

public class Employee {
    private String employeeId;
    private String employeeName;
    public String getEmployeeId() {
        return employeeId;
    }
    public void setEmployeeId(String employeeId) {
        this.employeeId = employeeId;
    }
    public String getEmployeeName() {
        return employeeName;
    }
}
```



```

        public void setEmployeeName(String employeeName) {
            this.employeeName = employeeName;
        }
    }
}

package com.journaldev.jsfBeans;

import java.util.ArrayList;
import java.util.List;

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

import com.journaldev.data.Employee;

@ManagedBean
@SessionScoped
public class ViewEmployeesManagedBean {
    private List<Employee> employees = new ArrayList<Employee>();

    public ViewEmployeesManagedBean() {

    }

    @PostConstruct
    public void populateEmployeeList() {
        for(int i = 1 ; i <= 10 ; i++){
            Employee emp = new Employee();
            emp.setEmployeeId(String.valueOf(i));
            emp.setEmployeeName("Employee#" + i);
            this.employees.add(emp);
        }
    }

    public List<Employee> getEmployees() {
        return employees;
    }

    public void setEmployees(List<Employee> employees) {
        this.employees = employees;
    }
}

```

Notice the use of JSF annotations and use of PostConstruct annotation to populate the list of employees.

```

<!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"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">
<p:outputLabel value="JournalDev - JSF2/Primefaces
Tutorial"></p:outputLabel>
<p:dataTable value="#{viewEmployeesManagedBean.employees}"
var="employee">
  <p:column>
    <f:facet name="header">
      <h:outputText value="Employee ID"></h:outputText>
    </f:facet>
    <h:outputText value="#{employee.employeeId}"></h:outputText>
  </p:column>
  <p:column>
    <f:facet name="header">
      <h:outputText value="Employee Name"></h:outputText>
    </f:facet>
    <h:outputText value="#{employee.employeeName}"></h:outputText>
  </p:column>
</p:dataTable>
</html>

```

Notice the use of dataTable element to create the table from the managed bean properties. PrimeFaces and JSF takes care of passing these to the view page for rendering. If you are from Servlet background, you can easily see that number of steps are cut down – in servlet environment, we first handle the request in servlet, create the model data, set it as attribute in request/session and then forward it to the JSP page to render the response.

```

<?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"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>JournalDev-PrimefacesWebApplication</display-name>
  <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>
  <context-param>

```

```

        <description>State saving method: 'client' or 'server'
        (=default). See JSF Specification 2.5.2</description>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>client</param-value>
    </context-param>
    <context-param>
        <param-
name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
        <param-value>resources.application</param-value>
    </context-param>
    <listener>
        <listener-
class>com.sun.faces.config.ConfigureListener</listener-class>
    </listener>
</web-app>

```

Notice that `javax.faces.webapp.FacesServlet` is the controller class, this is where we plugin JSF into our web application.

```

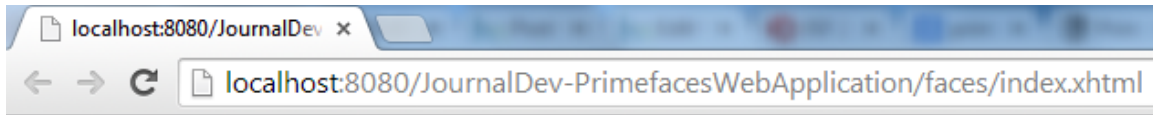
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
    version="2.2">

</faces-config>

```

This is where we provide JSF components configurations such as managed beans, global messages, custom view handlers and custom factory classes. Since we are using annotations and it's a simple project, there is no configuration done here, but we will see it's usage in future posts.

Now when you will run this, you will get output as shown in below image.



## JournalDev Tutorials

### JournalDev - JSF2/Primefaces Tutorial

Employee ID	Employee Name
1	Employee#1
2	Employee#2
3	Employee#3
4	Employee#4
5	Employee#5
6	Employee#6
7	Employee#7
8	Employee#8
9	Employee#9
10	Employee#10

## b. Convert into Maven

Maven is the most preferred way to manage the java projects build and dependencies, so here we will see how we can convert it to Maven. Eclipse IDE provide the option to convert your **Dynamic Web Project** into Maven. Maven will help you controlling and managing the required dependencies. Just right click on the created project and from configure menu select **Convert into Maven Project**.

Once you have changed your project into Maven, you have to add the required dependencies for making the project compilable by the Maven itself.

The supposed Maven XML that you gain once you've converted the application into Maven project and after adding the required libraries for JSF 2, Primefaces and other is:

```

<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/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>JournalDev-PrimefacesWebApplication</groupId>
  <artifactId>JournalDev-PrimefacesWebApplication</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.3</version>
        <configuration>
          <warSourceDirectory>webapp</warSourceDirectory>
          <failOnMissingWebXml>>false</failOnMissingWebXml>
        </configuration>
      </plugin>
    </plugins>
    <finalName>${project.artifactId}</finalName>
  </build>
  <repositories>
    <repository>
      <id>prime-repo</id>
      <name>PrimeFaces Maven Repository</name>
      <url>http://repository.primefaces.org</url>
      <layout>default</layout>
    </repository>
  </repositories>
  <dependencies>
    <!-- Servlet -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>
    <!-- Faces Implementation -->
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-impl</artifactId>
      <version>2.2.4</version>
    </dependency>
  </dependencies>

```

```

<!-- Faces Library -->
<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.2.4</version>
</dependency>
<!-- Primefaces Version 5 -->
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>5.0</version>
</dependency>
<!-- JSP Library -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.1</version>
</dependency>
<!-- JSTL Library -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.1.2</version>
</dependency>
</dependencies>
</project>

```

And by executing the **mvn clean package** against the project, you will get a WAR file ready for being deployed against any of the Java EE container. Just deploy and examine.

Here we introduced how you could use a JSF 2 / Primefaces for implementing a web application user interface. In that, we've used Eclipse IDE for achieving that, by creating a dynamic project followed by adding all of required libraries either those mandatory for JSF 2 implementation or those required for using Primefaces components. From next tutorial onwards, we will use Maven to create the project for our examples.

**You can download the source code of “PrimeFaces Hello World Project” from [here](#).**

## 2. Primefaces AccordionPanel Component Tutorial

As we've introduced previously about **Primefaces**, it's a leading UI library for JSF that contains set of ready-made components helping you building compelling User Interfaces.

As also, we've explained you how to get ready to use PrimeFaces in the [Primefaces 5 / JSF 2 Beginners Example Tutorial](#), so be sure that you go through it if you are new to PrimeFaces and JSF technologies.

The excellent reputation that **Primefaces** has gained is because of the dozens of UI components it provides. **AccordionPanel**, is one of those components and it's a container component that displays content in a stacked format.

This tutorial is intended to provide you the all required information and practices for dealing with the **AccordionPanel** component, even using of newly primefaces's paradigm, which consists of employing a special **JavaScript library object** for controlling the component itself.

### A. AccordionPanel Basic Info

It's important for any developer has been coming into leveraging any of the primefaces component, know the basic information for the component being used. The basic information gives the developer information like the tag that would be used in the Primefaces page, the component type, class, family and all of **renderer** information required in case a customizing is targeted.

Per Primefaces itself, the basic information of AccordionPanel are:

Tag	<b>accordionPanel</b>
Component Class	<b>org.primefaces.component.accordionpanel.Accordionpanel</b>
Component Type	<b>org.primefaces.component.AccordionPanel</b>
Component Family	<b>org.primefaces.component</b>
Renderer Type	<b>org.primefaces.component.AccordionPanelRenderer</b>
Renderer Class	<b>org.primefaces.component.accordionpanel.AccordionPanelRenderer</b>

## B. AccordionPanel Attributes

Unlike the basic information that you've got in the Basic info section, the attributes help the developer controlling the component by means of visibility, populating data, styling, triggering certain behaviors and others.

Per Primefaces, the all attributes are supported by AccordionPanel are:

Name	Default	Type	Description
id	null	String	Unique identifier of the component
rendered	true	boolean	Boolean value to specify the rendering of the component.
binding	null	Object	An EL expression that maps to a server side UIComponent instance in a backing bean.
activeIndex	false	String	Index of the active tab or a comma separated string of indexes when multiple mode is on.



Name	Default	Type	Description
style	null	String	Inline style of the container element.
styleClass	null	String	Style class of the container element.
onTabChange	null	String	Client side callback to invoke when an inactive tab is clicked.
onTabShow	null	String	Client side callback to invoke when a tab gets activated.
dynamic	false	Boolean	Defines the toggle mode.
cache	true	Boolean	Defines if activating a dynamic tab should load the contents from server again.
value	null	List	List to iterate to display dynamic number of tabs.
var	null	String	Name of iterator to use in a dynamic number of tabs.
multiple	false	Boolean	Controls multiple selection.
dir	ltr	String	Defines text direction, valid values are <i>ltr</i> and <i>rtl</i> .
prependId	true	Boolean	AccordionPanel is a naming container thus prepends its id to its children by default, a false value turns this behavior off except for dynamic tabs.
widgetVar	null	String	Name of the client side widget.

After that brief description about the AccordionPanel, it's the time to get started using it and knowing how it works and how could we leverage its attributes to serve the business case that we have. But before getting into, it's important to know that the tutorial being implemented has been evolved using the maven archetype. In case you are familiar with, just skip the next section toward implementing the business scenario directly.

## C. Create Project Using Eclipse IDE

You've already created your project using Eclipse IDE **Dynamic Web Project**, but from now you will learn basically a new way of creating a Maven Project through Eclipse IDE without the need to convert the Dynamic Web Project into Maven. It's an easy-peasy way, so just follow:

1. Make sure that you have installed Maven into your machine.
2. From your **Project Explorer** RightClick – Select New – Project.
3. Expand Maven node and select Maven Project and Click double Next.

4. From all of those templates that Eclipse provided for you, makes sure that you are selected the line beneath groupId: **org.apache.maven.archetypes** and the artifactId: **maven-archetype-webapp** and click next.
5. Enter your preferred groupId, artifactId, version and package and click finish.

**New Maven Project**

Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

Properties available from archetype:

Name	Value

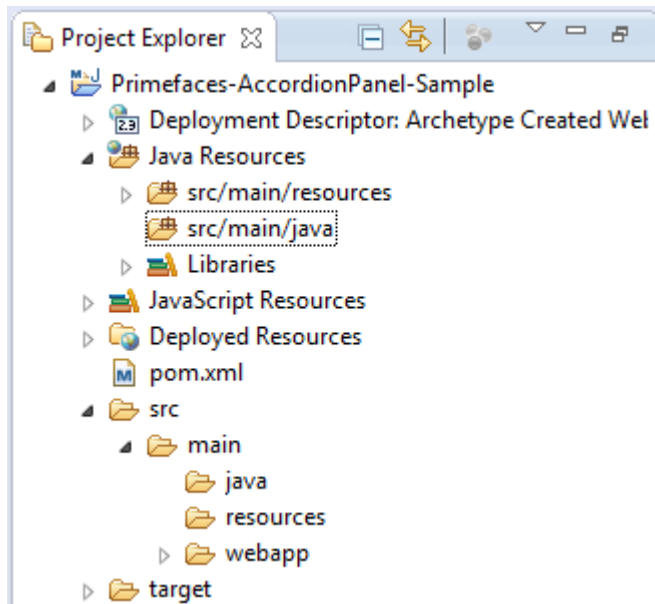
▶ Advanced

< Back   Next >   **Finish**   Cancel

Once you've finished the dialog, you are already got a Maven based project ready for your use. Before going forward into the next step, just makes sure that you have a **java** folder under your **Java Resources**, for doing that just do the following:

1. Right Click **Java Resources** that located under the created project.
2. Select **New – Source Folder**.
3. Browse your project.
4. Enter **src/main/java** inside Folder name.
5. Click finish.

Your final structure looks like below



## D. Setting up Required Files & Dependencies

As Maven promised, the all dependencies required for your project for making it runnable must be located into your maven configuration file and it's for sure the **pom.xml**.

For being the targeted application has used the primefaces library for which the jsf, jsp and servlet libraries are required, the form of the pom.xml should be:

```

<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.journaldev</groupId>
  <artifactId>Primefaces-AccordionPanel-Sample</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>Primefaces-AccordionPanel-Sample Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <repositories>
    <repository>
      <id>prime-repo</id>
      <name>PrimeFaces Maven Repository</name>
      <url>http://repository.primefaces.org</url>
      <layout>default</layout>
    </repository>
  </repositories>
  <dependencies>
    <!-- Servlet -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>
    <!-- Faces Implementation -->
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-impl</artifactId>
      <version>2.2.4</version>
    </dependency>
    <!-- Faces Library -->
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-api</artifactId>
      <version>2.2.4</version>
    </dependency>
    <!-- Primefaces Version 5 -->
    <dependency>
      <groupId>org.primefaces</groupId>
      <artifactId>primefaces</artifactId>
      <version>5.0</version>
    </dependency>
    <!-- JSP Library -->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>

```

```

        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>2.3.1</version>
    </dependency>
    <!-- JSTL Library -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.1.2</version>
    </dependency>
</dependencies>
</project>

```

As for being the application uses the jsf framework, a set of needed lines should be added for making that framework running. Starting from the jsf listener – that should help the web container configure the jsf framework’s components like jsf custom components, validators and converters that defined in the non-required **faces-config.xml** or by using the annotations – ending with those required files like jsf framework properties, faces servlet and its mapping.

The whole work that you are located in the web.xml for establishing a full functional jsf isn’t far from what you would see 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" metadata-complete="true">
    <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>*.xhtml</url-pattern>
    </servlet-mapping>
    <context-param>
        <description>State saving method: 'client' or 'server'
        (=default). See JSF Specification 2.5.2</description>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>client</param-value>
    </context-param>

```

```

        </context-param>
        <listener>
            <listener-
class>com.sun.faces.config.ConfigureListener</listener-class>
        </listener>
    </web-app>

```

You might want to ask, where is the **faces-config.xml** ? In fact and indeed, since the jsf 2, the faces-config.xml isn't longer used, given hundreds of annotations had covered the needed for defining the jsf component you may need. But for sure if you want to use it, it's applicable and you have just add the following line into your **web.xml** file.

```

<context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>
        /WEB-INF/faces-config.xml
    </param-value>
</context-param>

```

For now, you have established the jsf framework and the all libraries required for leveraging the Primefaces into your application. So, let's start developing an application uses the AccordionPanel component.

For now, almost you are finished the all required steps for using the Primefaces 5 – AccordionPanel component, but one step just remaining.

As we knew, jsf 2 has obsoleted using of JSP (Java Server Pages) as view library, it's now Facelets (XHTML Pages) that should comply with the XML well-formed principle.

Facelets library is already embedded in the jsf 2 dependencies and so, no need for adding a new libraries. For creating a new view using **Facelets** you shall follow the below steps:

1. Right Click on the **webapp** folder.
2. Select New – HTML File.
3. Name your file as **index** and makes sure that the extension is xhtml & Click Next.
4. Just finish the wizard.

By end of these steps, you will get an **index.xhtml** facelets page which will be updated accordingly for using the AccordionPanel. Just let's your page like as 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"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui">

</html>

```

## E. Getting Started With AccordionPanel

AccordionPanel consists of one or more tabs and each tab can group any content. Titles can also be defined with “title” facet. As a very simple using of AccordionPanel, you can consider the following lines as the minimal required code fragments for proper use of it.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

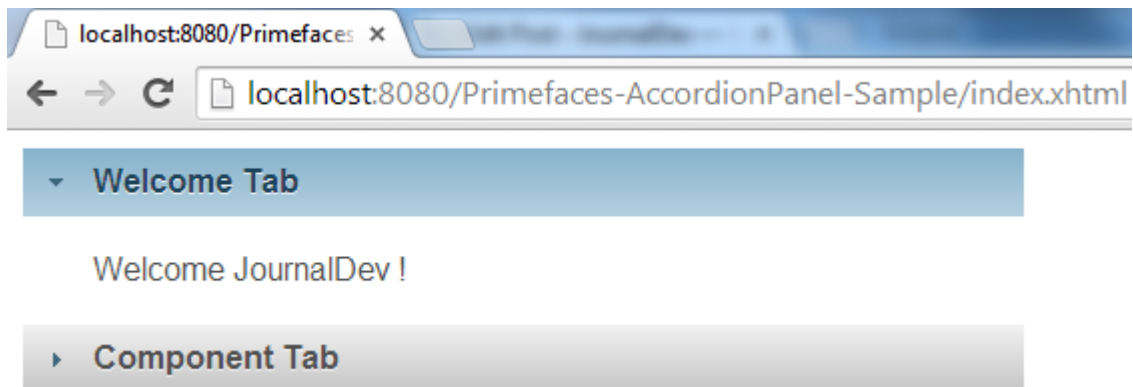
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui">
    <h:head>
        <script name="jquery/jquery.js" library="primefaces"></script>
    </h:head>
    <h:body>
        <div style="width: 500px;">
            <p:accordionPanel activeIndex="0">
                <p:tab title="Welcome Tab">
                    <p:outputLabel value="Welcome JournalDev
!"></p:outputLabel>
                </p:tab>
                <p:tab title="Component Tab">
                    <p:outputLabel value="AccordionPanel Component
!"></p:outputLabel>
            </p:accordionPanel>
        </div>
    </h:body>
</html>

```

```
        </p:tab>
    </p:accordionPanel>
</div>
</h:body>
</html>
```

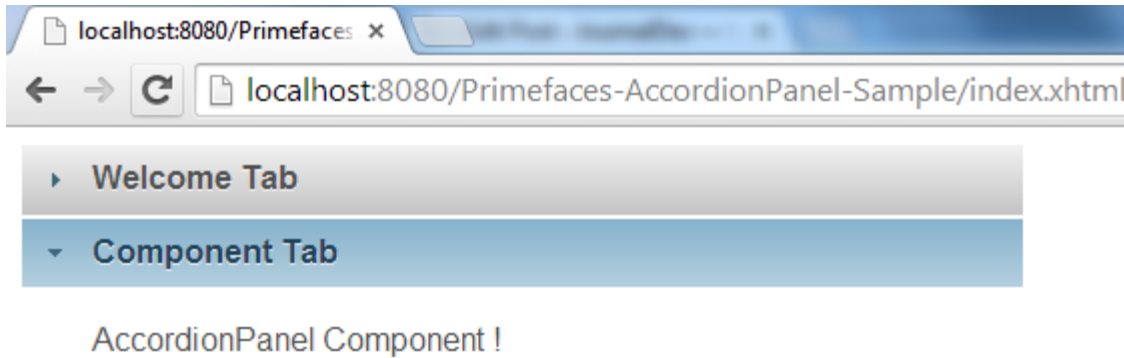
Let's analyze the above code for good understanding of it.

- Primefaces JavaScript library is mandatory line, because the all components within Primefaces have used that library for rendering styles and executing client side behaviors.
- JSF 2 library, provides you the <h:head/> & <h:body/> for representing the header & body normal tags that are defined in the HTML.
- AccordionPanel component has defined in the page for containing two Tabs indexed starting from zero. The activeIndex attribute used for defining which Tabs from those defined should be displayed actively when the page get displayed. See AccordionPanel attributes section above.



As you've seen, AccordionPanel displays two different tabs in their respective order, they're containing whatever content you wish. For being the activeIndex is zero by default, the first tab will be displayed once the page get shown. In case you've clicked on the Component Tab, the second tab will be displayed, while the Welcome Tab get closed.





If you've set `activeIndex` value equal to 1, `Component Tab` get displayed firstly.

## F. AccordionPanel – Dynamic Content Loading

`AccordionPanel` supports lazy loading of tab content, when `dynamic` attribute is set to `true`, only active tab contents will be rendered to the client side and clicking an inactive tab header will do an ajax request to load the tab contents.

This feature is useful to reduce bandwidth and speed up page loading time. By default activating a previously loaded dynamic tab does not initiate a request to load the contents again as tab is cached. To control this behavior use **cache** option.

For clarifying the dynamic concept, we've developed an `AccordionPanelManagedBean` that contains two **String** properties with their respective getter & setter.

For each tab within the `accordionPanel`, an `outputPanel` component has been defined and its value is associated with a one of those defined properties. In case the tab has been loaded, a print message should be printed. By default the loading of certain tab within `AccordionPanel` means that the contained components being rendered. The rendition process does affect calling of getter methods. Let's see the impact of using **dynamic** and **cache** attributes.

Firstly, in case they're never used, neither `dynamic` nor `cache`.

```

<!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"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui">
    <h:head>
        <script name="jquery/jquery.js" library="primefaces"></script>
    </h:head>
    <h:body>
        <div style="width: 500px;">
            <p:accordionPanel activeIndex="0">
                <p:tab title="Welcome Tab">
                    <p:outputLabel
value="#{accordionPanelManagedBean.journalMessage}"></p:outputLabel>
                </p:tab>
                <p:tab title="Component Tab">
                    <p:outputLabel
value="#{accordionPanelManagedBean.componentMessage}"></p:outputLabel>
                </p:tab>
            </p:accordionPanel>
        </div>
    </h:body>
</html>
package com.journaldev.primefaces.beans;

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

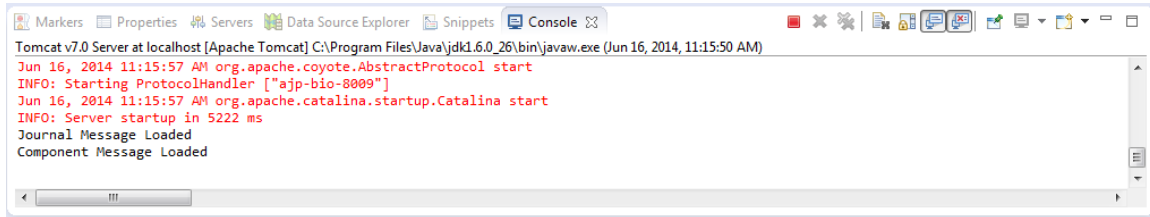
@ManagedBean
@SessionScoped
public class AccordionPanelManagedBean {

    private String journalMessage = "Welcome JournalDev !";
    private String componentMessage = "AccordionPanel Component !";

    public String getJournalMessage() {
        System.out.println("Journal Message Loaded");
        return journalMessage;
    }
    public void setJournalMessage(String journalMessage) {
        this.journalMessage = journalMessage;
    }
    public String getComponentMessage() {
        System.out.println("Component Message Loaded");
        return componentMessage;
    }
    public void setComponentMessage(String componentMessage) {
        this.componentMessage = componentMessage;
    }
}

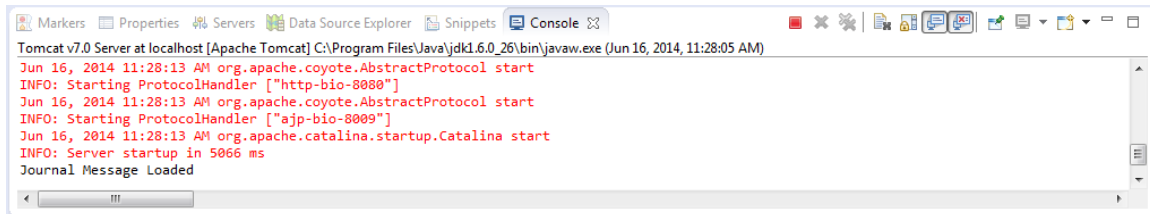
```

Where the resulted execution should be like this.

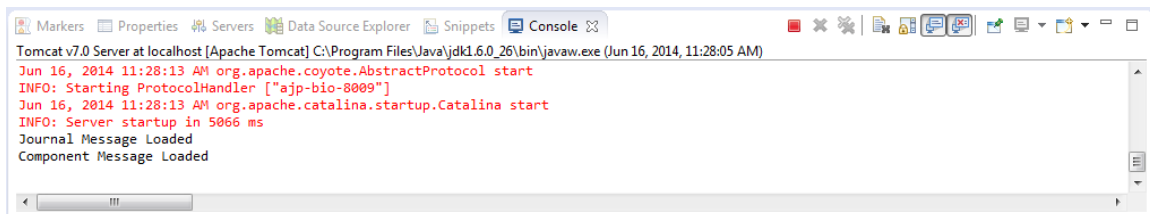


So, the two tabs have been loaded even the active tab is set zero, that's because the **dynamic** value was set false. At the other hand the **cache** attribute is set true, so any additional clicking on the tabs will not cause a new loading to be initiated. Even if you've changed the value **cache** to be true, without using of **dynamic** no changes might be noticed.

By adding the **dynamic** attribute into accordionPanel component with true value and keeping the **cache** with its default value. The result would be

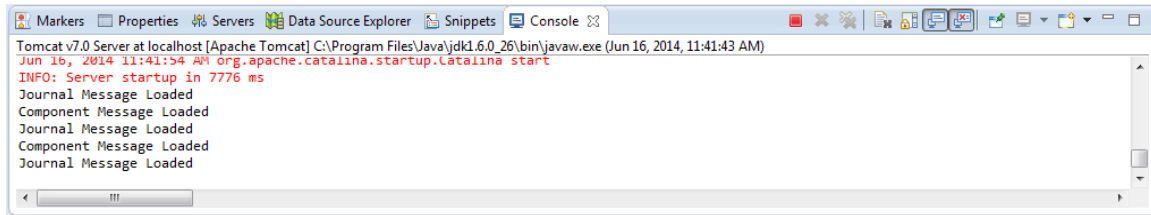


Once the view get displayed. Now, when clicking the component tab the result should be



As you've noticed, component tab isn't loaded until you cause that loading by clicking on. Any next coming clicking on that tabs will not cause the loading anymore, thus, no new messages may be seen. If you would to see the messages at every time you've clicked, you shall override the default value of **cache** to be false with retaining the existence of **dynamic**.

The result of such that change should be



That's expected, set cache value to be false, meaning that at every click, the content will be loaded instantly.

## G. AccordionPanel – Ajax Behavior Events

One of the most benefits that the jsf come with, is the ajax. Ajax has built into jsf lifecycle inherently, so for ajaxifying any component you would, you've just added the original **f:ajax** behavior into its tag. Primefaces 5 isn't exceptional case, as it's provided its own ajax behaviors using its own tag. For ajaxifying any of Primefaces 5 components that you would, you just need to provide **p:ajax** tag inside the component tag.

AccordionPanel is one of the primefaces component that leverage the concept of ajax, it provides you **tabChange** as a target behavior event that might be ajaxified. That event is executed when a tab is toggled.

For all ajax behaviors in Primefaces, a listener should be defined in the managed bean for listening the event behavior fired.

You can provide an **tabChange** ajax event by using the **p:ajax** while the listening could be done through defining a public void method that takes a **TabChangeEvent** as an argument. Just look at the followed example.

```
<!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"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui">
    <h:head>
        <script name="jquery/jquery.js" library="primefaces"></script>
    </h:head>
```

```

    <h:body>
        <h:form>
            <div style="width: 500px;">
                <p:accordionPanel activeIndex="0">
                    <p:ajax event="tabChange"
listener="#{accordionPanelManagedBean.onChange}"></p:ajax>
                    <p:tab title="Welcome Tab">
                        <p:outputLabel
value="#{accordionPanelManagedBean.journalMessage}"></p:outputLabel>
                    </p:tab>
                    <p:tab title="Component Tab">
                        <p:outputLabel
value="#{accordionPanelManagedBean.componentMessage}"></p:outputLabel>
                    </p:tab>
                </p:accordionPanel>
            </div>
        </h:form>
    </h:body>
</html>
package com.journaldev.primefaces.beans;

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

import org.primefaces.event.TabChangeEvent;

@ManagedBean
@SessionScoped
public class AccordionPanelManagedBean {

    private String journalMessage = "Welcome JournalDev !";
    private String componentMessage = "AccordionPanel Component !";

    public String getJournalMessage() {
        System.out.println("Journal Message Loaded");
        return journalMessage;
    }

    public void setJournalMessage(String journalMessage) {
        this.journalMessage = journalMessage;
    }

    public String getComponentMessage() {
        System.out.println("Component Message Loaded");
        return componentMessage;
    }

    public void setComponentMessage(String componentMessage) {
        this.componentMessage = componentMessage;
    }

    public void onChange(TabChangeEvent event) {

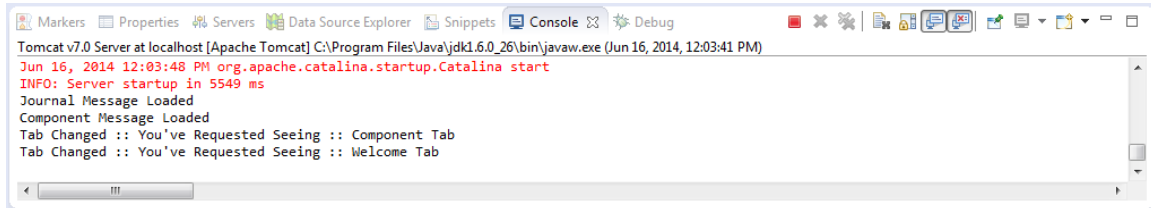
```

```

        System.out.println("Tab Changed :: You've Requested Seeing :: 
"+ event.getTab().getTitle());
    }
}

```

Where the result of execution should be like below



Just one note should be noticed here, is that the ajax is a normal event that handled by the jsf lifecycle, thus it's mandatory for you to put your component within an **h:form** and if you were missed it up, your ajax won't fired and your listener isn't invoked.

## H. AccordionPanel – Client Side API

Primefaces featured a new client side JavaScript API that's used for make Primefaces' components full functional plus the ability of controlling them through the **PF** object. **PF is an implicit JavaScript object that provided by the Primefaces JavaScript API, it accepts a WidgetVar variable as a component argument as allowing the developer invoke component's methods through it.** Before Pimefaces 5, it was applicable for you to invoke the methods for a certain component through using the **WidgetVar** itself. Since Primefaces 4, using of **WidgetVar** variable directly has been deprecated and in the version 5 it's removed totally.

Primefaces JavaScript library provides you two methods can be invoked against AccordionPanel component; **select(index)** and **unselect(index)**. By passing the accordionPanel widgetVar into PF object, those provided methods are accessible there.

Look below, an example provided for AccordionPanel get controlled by using JavaScript library.

```

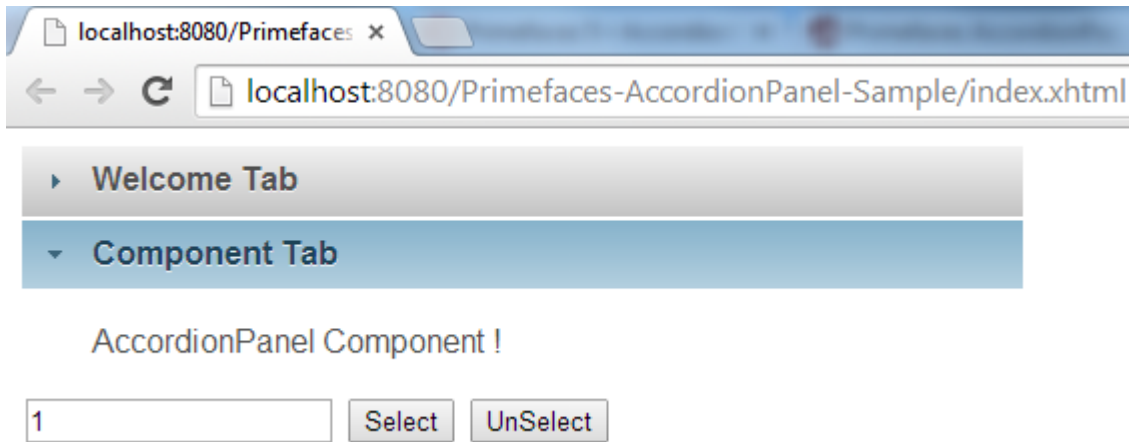
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui">
    <h:head>
        <script name="jquery/jquery.js" library="primefaces"></script>
        <script>
            alert(PF);
            function doSelect(index){
                PF('accordionPanel').select(index);
            }

            function doUnSelect(index){
                PF('accordionPanel').unselect(index);
            }
        </script>
    </h:head>
    <h:body>
        <h:form>
            <div style="width: 500px;">
                <p:accordionPanel widgetVar="accordionPanel"
activeIndex="0">
                    <p:ajax event="tabChange"
listener="#{accordionPanelManagedBean.onChange}"></p:ajax>
                    <p:tab title="Welcome Tab">
                        <p:outputLabel
value="#{accordionPanelManagedBean.journalMessage}"></p:outputLabel>
                    </p:tab>
                    <p:tab title="Component Tab">
                        <p:outputLabel
value="#{accordionPanelManagedBean.componentMessage}"></p:outputLabel>
                    </p:tab>
                </p:accordionPanel>
                <input id="index" type="text"/>
                <input id="Select" value="Select" type="button"
onclick="doSelect(document.getElementById('index').value);false"/>
                <input id="UnSelect" value="UnSelect" type="button"
onclick="doUnSelect(document.getElementById('index').value);false"/>
            </div>
        </h:form>
    </h:body>
</html>

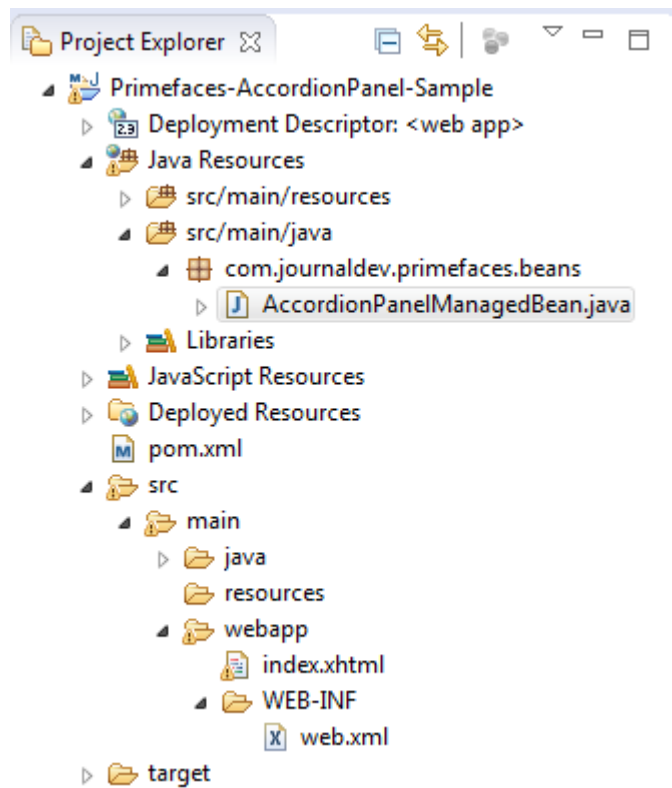
```

As being seen, we have two JavaScript actions invoke the select and unselect methods against AccordionPanel component.



Just put inside the number index you want, and see the the component does respond for your client side actions.

After finishing the tutorial, it's good for you to see the all structure that the project reached into.





Primefaces 5 provides you dozens of ready made UI controls, one of them is the AccordionPanel. This tutorial has introduced the AccordionPanel component with all of its capabilities that might need when coming into using it. Download the sample project from below link and play around with it to learn more.

**You can download the source code of “PrimeFaces AccordionPanel Project” from [here](#).**

# 3. Primefaces Spring & Hibernate Integration

Integration between frameworks is a complex mission and mostly it needs a lot of time to achieve. We've discussed a [Primefaces](#), [Spring](#) and [Hibernate](#) frameworks in a separate tutorials, but this time we will show you how can integrate all of them to create a layered (tiered) application.

Layered (tiered) application is a popular design that most of the enterprise applications are aligned with. In which:

- Primefaces framework will be used for handling all UI concerns and verify client's inputs.
- [Hibernate framework](#) will be used for communicating your own persistence store that probably is a MySQL database.
- [Spring framework](#) will be used to glue between all of these frameworks.

This tutorial intended for implementing a layered application using all of these listed frameworks.

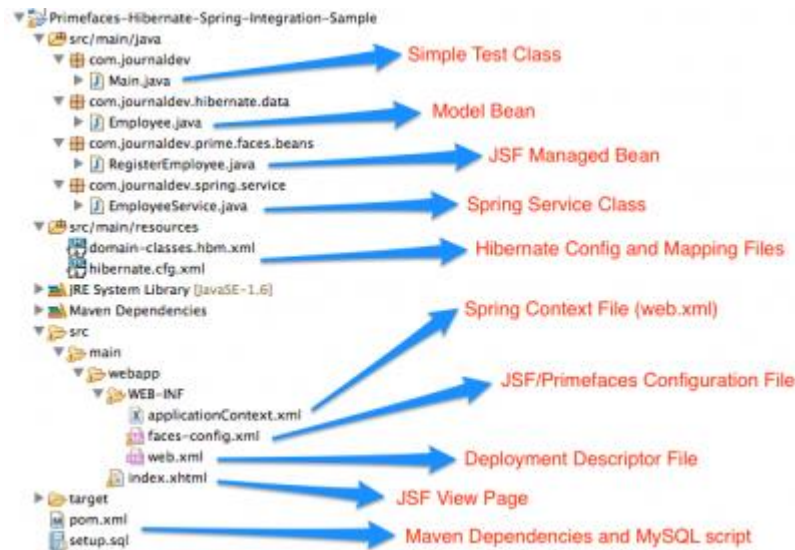
## A. Required Tools

Before getting started delve into, let's see the required tools that you would need for:

- Eclipse Kepler 4.3.
- Hibernate 3.x.
- Spring 4.x.
- Primefaces 5.x.
- JDK 1.6+.
- MySQL 5.x.

## B. Primefaces Spring Hibernate Project Structure

Our final project structure will look like below image, we will go through each of the components one by one.



### a. Create Database Employee Table

MySQL database would be used for retaining all of employees instances/records. Used Employee Table looks like below:

Table Name:  Schema: **journaldev**

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
EMP_ID	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
EMP_NAME	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
EMP_HIRE_DATE	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
EMP_SALARY	DECIMAL(11,4)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Column Name:   
Collation:   
Comments:

Data Type:   
Default:   
☐ Primary Key   ☐ Not Null   ☐ Unique  
☐ Binary   ☐ Unsigned   ☐ Zero Fill  
☐ Auto Increment

Columns   **Indexes**   Foreign Keys   Triggers   Partitioning   Options

Also, find below its SQL create-script:

```
CREATE TABLE `employee` (
  `EMP_ID` int(11) NOT NULL AUTO_INCREMENT,
  `EMP_NAME` varchar(45) DEFAULT NULL,
  `EMP_HIRE_DATE` datetime DEFAULT NULL,
  `EMP_SALARY` decimal(11,4) DEFAULT NULL,
  PRIMARY KEY (`EMP_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

## b. Create Employee Domain Class

After we created an Employee Table, it's a proper time to get a look at how Employee class would look like:

```
package com.journaldev.hibernate.data;

import java.util.Date;

public class Employee {
    private long employeeId;
    private String employeeName;
    private Date employeeHireDate;
    private double employeeSalary;

    public long getEmployeeId() {
        return employeeId;
    }
}
```

```

    public void setEmployeeId(long employeeId) {
        this.employeeId = employeeId;
    }

    public String getEmployeeName() {
        return employeeName;
    }

    public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
    }

    public Date getEmployeeHireDate() {
        return employeeHireDate;
    }

    public void setEmployeeHireDate(Date employeeHireDate) {
        this.employeeHireDate = employeeHireDate;
    }

    public double getEmployeeSalary() {
        return employeeSalary;
    }

    public void setEmployeeSalary(double employeeSalary) {
        this.employeeSalary = employeeSalary;
    }
}

```

## C. Setting Up Maven

Maven is a build tool, it's used mainly for managing project dependencies. So no need for downloading JARs and appending them into your project as you did normally. **MySQL JDBC driver, hibernate core, Spring core framework, Primefaces** and many libraries are completely downloaded by Maven and they will be located into your classpath automatically.

```

<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.journaldev</groupId>
    <artifactId>Primefaces-Hibernate-Spring-Integration-
Sample</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>

```

```

    <name>Primefaces-Hibernate-Spring-Integration-Sample Maven
Webapp</name>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <url>http://maven.apache.org</url>
    <repositories>
        <repository>
            <id>prime-repo</id>
            <name>PrimeFaces Maven Repository</name>
            <url>http://repository.primefaces.org</url>
            <layout>default</layout>
        </repository>
    </repositories>
    <dependencies>
        <!-- Servlet -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
        <!-- Faces Implementation -->
        <dependency>
            <groupId>com.sun.faces</groupId>
            <artifactId>jsf-impl</artifactId>
            <version>2.2.4</version>
        </dependency>
        <!-- Faces Library -->
        <dependency>
            <groupId>com.sun.faces</groupId>
            <artifactId>jsf-api</artifactId>
            <version>2.2.4</version>
        </dependency>
        <!-- Primefaces Version 5 -->
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>5.0</version>
        </dependency>
        <!-- JSP Library -->
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>

```

```

        <version>2.3.1</version>
    </dependency>
    <!-- JSTL Library -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.1.2</version>
    </dependency>
    <!-- Primefaces Theme Library -->
    <dependency>
        <groupId>org.primefaces.themes</groupId>
        <artifactId>blitzer</artifactId>
        <version>1.0.10</version>
    </dependency>
    <!-- Hibernate library -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.3.6.Final</version>
    </dependency>
    <!-- MySQL driver connector library -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.31</version>
    </dependency>
</dependencies>
</project>

```

## D. Setting Up Hibernate

Hibernate is a standard Object Relational Mapping (ORM) solution, it's used for mapping your object domain into relational tabular formula. Setup of hibernate process does comprise all of the below steps:

- Specify all relevant database information like driver, JDBC URL, hibernate dialect and hibernate session context in a hibernate configuration file, mainly using **hibernate.cfg.xml**. Dialect will be used by hibernate implementation itself for make sure the execution of mapping process is done effectively. This file should be located under project's src/main/resources folder.
- Specify hibernate's mapping file. Mapping file will contains all of mapping information, like objects-tables, attributes-columns and associations-relations, **domain-classes.hbm.xml** file is mainly used for

- this purpose. This file should be located under project's src/main/resources folder so that it's in the classpath of the application.
- It's important to say that some of modifications would be happened upon these files below when we're going to use spring.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Database connection settings -->
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/journaldev</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>

        <!-- SQL dialect -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <!-- Specify session context -->
        <property
name="hibernate.current_session_context_class">org.hibernate.context.internal.ThreadLocalSessionContext</property>
        <!-- Show SQL -->
        <property name="hibernate.show_sql">true</property>
        <!-- Referring Mapping File -->
        <mapping resource="domain-classes.hbm.xml"/>
    </session-factory>

</hibernate-configuration>

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
    <class name="com.journaldev.hibernate.data.Employee"
table="employee">
        <id name="employeeId" column="EMP_ID" type="long">
            <generator class="native" />
        </id>
        <property name="employeeName" column="EMP_NAME" type="string"/>
    </class>
</hibernate-mapping>
```



```

        <property name="employeeHireDate" column="EMP_HIRE_DATE"
type="date"/>
        <property name="employeeSalary" column="EMP_SALARY"
type="double"/>
    </class>
</hibernate-mapping>

```

## E. Test an Application

Till now, we've created an Eclipse Web project configured with required dependencies, created database Employee Table and created hibernate framework accompanies.

Before going far away with Spring integration and developing a Primefaces UI form, let's see how can we use a simple Java Application for getting Employee instance saved against our own database. Given Java Application would help us identifying the benefits we'll get especially when it comes to use a Spring framework later on.

```

package com.journaldev;

import java.util.Date;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import com.journaldev.hibernate.data.Employee;

public class Main {
    public static void main(String [] args){
        // Create a configuration instance
        Configuration configuration = new Configuration();
        // Provide configuration file
        configuration.configure("hibernate.cfg.xml");
        // Build a SessionFactory
        SessionFactory factory = configuration.buildSessionFactory(new
StandardServiceRegistryBuilder().configure().build());
        // Get current session, current session is already associated
with Thread
        Session session = factory.getCurrentSession();
        // Begin transaction, if you would like save your instances,
your calling of save must be associated with a transaction
        session.getTransaction().begin();
        // Create employee
        Employee emp = new Employee();

```

```

emp.setEmployeeName("Peter Jousha");
emp.setEmployeeSalary(2000);
emp.setEmployeeHireDate(new Date());
// Save
session.save(emp);
// Commit, calling of commit will cause save an instance of
employee
session.getTransaction().commit();
}
}

```

	EMP_ID	EMP_NAME	EMP_HIRE_DATE	EMP_SALARY
▶	18	Peter Jousha	2014-08-21 00:00:00	2000
*	NULL	NULL	NULL	NULL

Here's detailed clarifications for the above code:

- Hibernate requires a defined context for make your acquired session affected. Standard Java Application context can be achieved by providing hibernate's attribute **hibernate.current\_session\_context\_class**. Value of **org.hibernate.context.internal.ThreadLocalSessionContext** will be binded the context to the current executed thread. That's mean, if you've invoked any type of CRUD operations against **session** object within an active Transaction, they will be executing into your own database once the Transaction has committed. In our case, an new employee instance has been saved. If you've used hibernate 3, this property should be **thread** instead of using **ThreadLocalSessionContext**.
- Hibernate 4 is used for Testing purpose, this version of hibernate isn't applicable when it comes to integrate with Spring 4. To integrate with Spring 4, you've requested to use Hibernate 3.
- Using of latest version of hibernate requires you to use **StandardServiceRegistryBuilder** to build **SessionFactory**.

## F. Setting Up Spring

Spring is a comprehensive framework, it's used mainly for Inversion of Control (IoC) which consider the more general category of the well-known concept [Dependency Injection](#).

However, a provided simple Java Application keep you capable of getting your Employee instances saved against your own database, but typically, this isn't the way that most of applications use to configure their own hibernate persistence layer.

Using of Spring will help you avoiding all creating and associating objects stuffs. Creating required objects, associating others are mainly a Spring job. Following are Spring context configuration file, updated hibernate configuration, updated Maven pom.xml and our deployment descriptor file. Let's see how can we configure all of these to make a proper use of Spring.

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

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.2.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">
    <!-- Enable Spring Annotation Configuration -->
    <context:annotation-config />
    <!-- Scan for all of Spring components such as Spring Service -->
    <context:component-scan base-
package="com.journaldev.spring.service"></context:component-scan>
    <!-- Create Data Source bean -->
    <bean id="dataSource"
class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"
/>
        <property name="url"
value="jdbc:mysql://localhost:3306/journaldev" />
```

```

        <property name="username" value="root" />
        <property name="password" value="root" />
    </bean>
    <!-- Define SessionFactory bean -->
    <bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="mappingResources">
            <list>
                <value>domain-classes.hbm.xml</value>
            </list>
        </property>
        <property name="configLocation">
            <value>classpath:hibernate.cfg.xml</value>
        </property>
    </bean>
    <!-- Transaction Manager -->
    <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>
    <!-- Detect @Transactional Annotation -->
    <tx:annotation-driven transaction-manager="transactionManager" />
</beans>

```

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- SQL dialect -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    </session-factory>
</hibernate-configuration>

```

```

<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.journaldev</groupId>
    <artifactId>Primefaces-Hibernate-Spring-Integration-
Sample</artifactId>

```

```

    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>Primefaces-Hibernate-Spring-Integration-Sample Maven
Webapp</name>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <url>http://maven.apache.org</url>
    <repositories>
        <repository>
            <id>prime-repo</id>
            <name>PrimeFaces Maven Repository</name>
            <url>http://repository.primefaces.org</url>
            <layout>default</layout>
        </repository>
    </repositories>
    <dependencies>
        <!-- Servlet -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
        <!-- Faces Implementation -->
        <dependency>
            <groupId>com.sun.faces</groupId>
            <artifactId>jsf-impl</artifactId>
            <version>2.2.4</version>
        </dependency>
        <!-- Faces Library -->
        <dependency>
            <groupId>com.sun.faces</groupId>
            <artifactId>jsf-api</artifactId>
            <version>2.2.4</version>
        </dependency>
        <!-- Primefaces Version 5 -->
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>5.0</version>
        </dependency>
        <!-- JSP Library -->
        <dependency>

```

```

        <groupId>javax.servlet.jsp</groupId>
        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>2.3.1</version>
    </dependency>
    <!-- JSTL Library -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.1.2</version>
    </dependency>
    <!-- Primefaces Theme Library -->
    <dependency>
        <groupId>org.primefaces.themes</groupId>
        <artifactId>blitzer</artifactId>
        <version>1.0.10</version>
    </dependency>
    <!-- Hibernate library -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>3.6.10.Final</version>
    </dependency>
    <!-- MySQL driver connector library -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.31</version>
    </dependency>
    <!-- Spring ORM -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>4.0.3.RELEASE</version>
    </dependency>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>4.0.3.RELEASE</version>
    </dependency>
    <!-- Required By Hibernate -->
    <dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>1.4</version>
    </dependency>
    <dependency>
        <groupId>javassist</groupId>
        <artifactId>javassist</artifactId>
        <version>3.12.1.GA</version>
    </dependency>
</dependencies>
</project>

```

```

<?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" metadata-complete="true">
  <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>*.xhtml</url-pattern>
  </servlet-mapping>
  <context-param>
    <description>State saving method: 'client' or 'server'
    (=default). See JSF Specification 2.5.2</description>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
  </context-param>
  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>
    <listener>
    <listener-
class>com.sun.faces.config.ConfigureListener</listener-class>
    </listener>
  </web-app>

```

Here's detailed explanation for the code given above:

- We've created a SessionFactory by instantiating a Spring Bean called **sessionFactory**. Instantiating of SessionFactory does require passing an instant of data source instance, passing mapping files (domain-classes.hbm.xml as is), passing all required Hibernate properties via using of hibernate.cfg.xml. As you've noticed, hibernate.cfg.xml doesn't contain database information, cause it's defined instantly – **Data Source Bean** – and no need for hibernate session context, cause it's enriched by Apache Tomcat. Even Apache Tomcat isn't a managed server, but it contains facilities that make help create a contextual session.

- Transaction Manager will help you eliminate using a snippet of code like **session.getTransaction().begin()** and **commit()**. **@Transactional** annotation will be used alternatively. That's mean, executing of any Spring Service's methods annotated with **@Transactional** will be done in a Transactional manner. In case you've called a CRUD operation against your **session** within a Transactional scope like **session.save()**, it will be executing directly into your own database at the end of called method. That's called Transaction Demarcation.
- You can define your own Spring Services by using **@Component**. That will be scanned automatically.
- A new libraries are added into our **pom.xml** maven dependencies file, **common-dbcp** and **javassist** are required by hibernate 3. If you've noticed, these libraries aren't required for Hibernate 4.
- It's mandatory to add Spring context loader listener for your web.xml file. This listener required an **applicationContext.xml** to be defined underneath of **WEB-INF/** folder. This is the default location and name for Spring configuration context file. In case you would to change its location and name you must add below snippet of code provided with required path.

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/SpringContext.xml</param-value>
</context-param>
```

## G. Spring EmployeeService

In a layered application like what we're doing here, all of business operations must be achieved by services. Spring provides your ability to define your own services that would contain your own business rules. EmployeeService would contain the required business for create an Employee.

```
package com.journaldev.spring.service;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import com.journaldev.hibernate.data.Employee;
```



```

@Component
public class EmployeeService {
    @Autowired
    private SessionFactory sessionFactory;

    public SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Transactional
    public void register(Employee emp){
        // Acquire session
        Session session = sessionFactory.getCurrentSession();
        // Save employee, saving behavior get done in a transactional
manner
        session.save(emp);
    }
}

```

Here's detailed explanation for the above code:

- EmployeeService is a Spring service, @Component annotation is used for defining Spring Service. By default, Spring will scan your mentioned package(s) for locating your service based on context:component-scan Tag.
- @Autowired will help you get required instances injected. That's Dependency Injection or IoC (Inversion of Control) concept. It's important concept, it means that instead of allowing the developer controlling the process of creating instances and making required associations as well. It makes all of these creation and association in behind seen. That is an incredible power of Spring. @Autowired is used for injecting one instance of SessionFactory, if you're worry about performance issue, you can define your SessionFactory bean as a singleton scope. For complete information about autowiring, read [Spring autowiring example](#).
- @Transactional annotation is used for Transaction Demarcation purpose. Transaction demarcation is used for associating your contextual session with an active Transaction. That will cause a CRUD operation to get executed against your own database. You should go through [Spring Declarative Transaction Management Example](#).

# H. Primefaces Managed Bean – RegisterEmployee

Managed Bean is a JSF facility, and it's used for handling all required User Interface validations. In a layered application, Managed Bean is used for invoking Business services. You may be wondering once you know that it's applicable for you to inject EmployeeService Spring bean into your own Managed Bean. That becomes true if you're used @ManagedProperty annotation.

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
    version="2.2">
<application>
    <el-
resolver>org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-
resolver>
</application>
</faces-config>
```

```
package com.journaldev.prime.faces.beans;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;

import com.journaldev.hibernate.data.Employee;
import com.journaldev.spring.service.EmployeeService;

@ManagedBean
@SessionScoped
public class RegisterEmployee {

    @ManagedProperty("#{employeeService}")
    private EmployeeService employeeService;

    private Employee employee = new Employee();

    public EmployeeService getEmployeeService() {
        return employeeService;
    }
}
```

```

public void setEmployeeService(EmployeeService employeeService) {
    this.employeeService = employeeService;
}

public Employee getEmployee() {
    return employee;
}

public void setEmployee(Employee employee) {
    this.employee = employee;
}

public String register() {
    // Calling Business Service
    employeeService.register(employee);
    // Add message
    FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage("The Employee
"+this.employee.getEmployeeName()+" Is Registered Successfully"));
    return "";
}
}

```

Here's detailed explanation for the above code:

- RegisterEmployee Managed Bean is developed with using of @ManagedProperty annotation that will help you get a Spring EmployeeService instance injected. That association won't be applicable if you don't provide a special **faces-config.xml** file that contains a newly added Spring's el-resolver.
- Primefaces UI form will help you gather all required information about registered employee.
- Register action will ask EmployeeService saving given employee instance.

```

<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:p="http://primefaces.org/ui">
<h:head>
    <script name="jquery/jquery.js" library="primefaces"></script>
    <title>Register Employee</title>
</h:head>
<h:form>
    <p:growl id="messages"></p:growl>
    <p:panelGrid columns="2">
        <p:outputLabel value="Enter Employee Name:"></p:outputLabel>

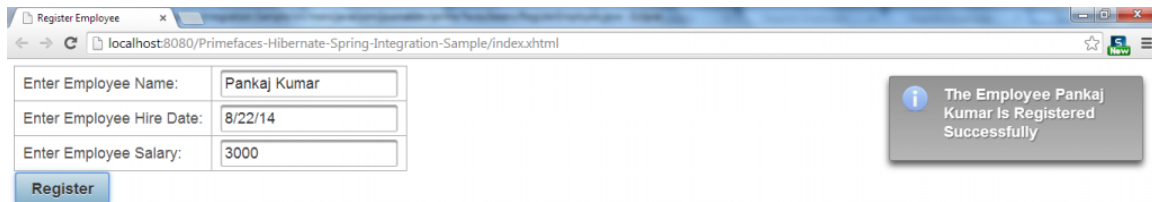
```

```

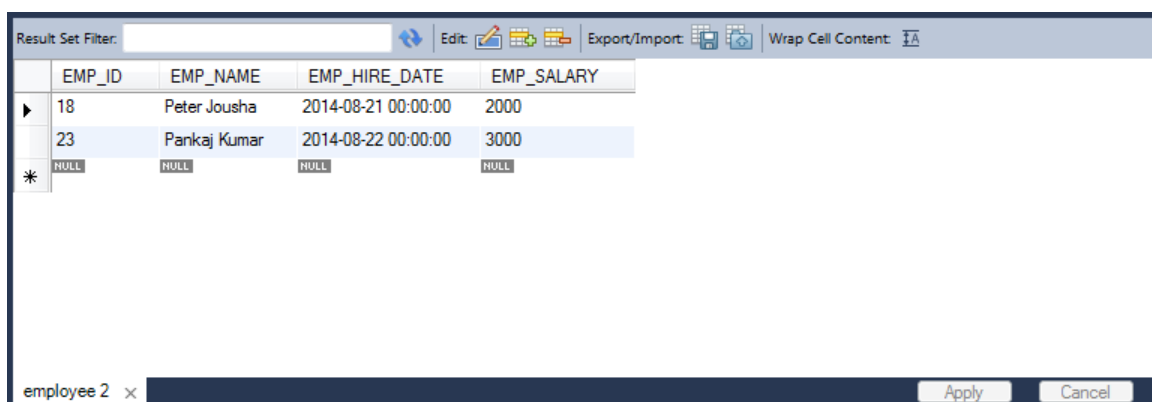
        <p:inputText
value="#{registerEmployee.employee.employeeName}"></p:inputText>
        <p:outputLabel value="Enter Employee Hire
Date:"></p:outputLabel>
        <p:calendar
value="#{registerEmployee.employee.employeeHireDate}"></p:calendar>
        <p:outputLabel value="Enter Employee Salary:"></p:outputLabel>
        <p:inputText
value="#{registerEmployee.employee.employeeSalary}"></p:inputText>
    </p:panelGrid>
    <p:commandButton value="Register"
action="#{registerEmployee.register}"
update="messages"></p:commandButton>
</h:form>
</html>

```

Primefaces - Register Form - Employee Registered - Notification Message



The screenshot shows a web browser window with the URL `localhost:8080/Primefaces-Hibernate-Spring-Integration-Sample/index.xhtml`. The page contains a registration form with three input fields: "Enter Employee Name:" (containing "Pankaj Kumar"), "Enter Employee Hire Date:" (containing "8/22/14"), and "Enter Employee Salary:" (containing "3000"). Below these fields is a "Register" button. To the right of the form, a notification message box displays: "The Employee Pankaj Kumar is Registered Successfully".



The screenshot shows a database table with the following columns: EMP\_ID, EMP\_NAME, EMP\_HIRE\_DATE, and EMP\_SALARY. The table contains two records:

EMP_ID	EMP_NAME	EMP_HIRE_DATE	EMP_SALARY
18	Peter Jousha	2014-08-21 00:00:00	2000
23	Pankaj Kumar	2014-08-22 00:00:00	3000

Below the table, there is a row with the value "\*" in the EMP\_ID column and "NULL" in the other columns. The table is titled "employee 2" and has "Apply" and "Cancel" buttons at the bottom right.

Hibernate integration with Spring and Primefaces is a popular development task. This tutorial guides you thoroughly to get Hibernate integrated with Spring and Primefaces that would lead you into getting an employee persisted against your database. Some technical details are mentioned intentionally. Contribute us by commenting below and find the source code for downloading purpose.

**You can download the source code of “PrimeFaces Spring Hibernate Integration Project” from [here](#).**

## 4. Primefaces, Spring 4 with JPA (Hibernate 4/EclipseLink) Example

**Java Persistence API** is a standard specification. It provides a persistence model that's implemented by different numerous of implementer.

Hibernate & EclipseLink are two most popular implementations used for persisting given business model against some sort of persistence store like relational database. As such, this tutorial will provide you a full-fledged example containing all required configuration steps to develop a layered application that uses:

- Primefaces components to develop a compelling User Interface that aimed to handle user's interactions and verify user's inputs.
- Hibernate/EclipseLink implementations to develop an Object/Relational Mapping beneath JPA umbrella.
- Spring framework as a kind of glue that get everything attached each together.

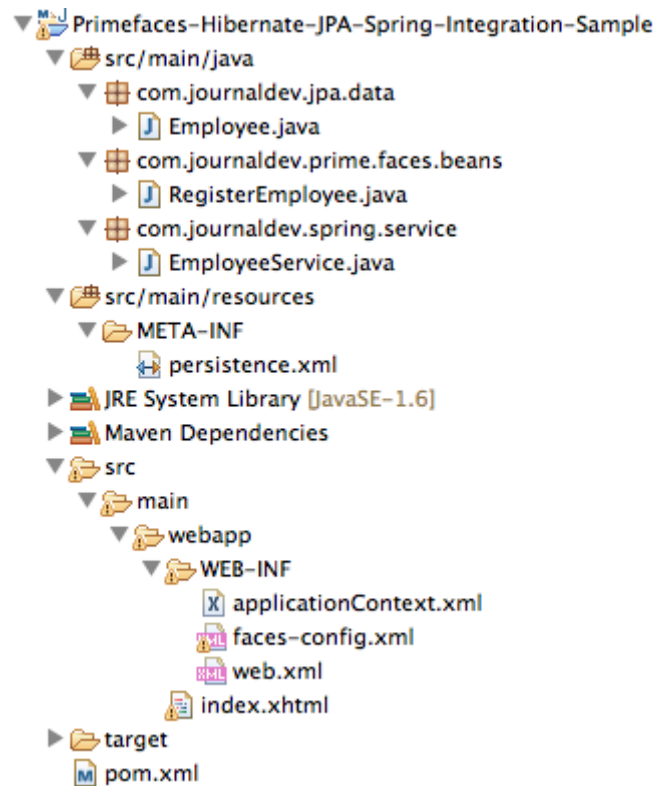
We've discussed before using of Hibernate ORM for persisting given domain classes in a multiple tutorials but today we will use only JPA based configurations. JPA specification does its bootstrap in a different way. In hibernate we've bootstrapped our application by using hibernate.cfg.xml file, but JPA doesn't specify such that file. As such, JPA provides another way of configuration, it's using of persistence.xml file which located within your classpath and under META-INF folder. Let's see how we can use both of Hibernate and EclipseLink for implementing a single registration form.

## Required Tools

Before proceeding far away, you must prepare your environments that should contain for:

- JDK 1.6+.
- Eclipse Kepler 4.3.
- Hibernate 4.3.6.Final.
- Spring 4.0.3.RELEASE.
- EclipseLink 2.5.0-RC1
- Maven Build Tool
- MySQL 5.x.

## A. Final Project Structure



### a. Database Tables

We have Employee table in our MySQL database, you can use below script to create it.

Table Name:  Schema: **journaldev**

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
EMP_ID	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
EMP_NAME	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
EMP_HIRE_DATE	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
EMP_SALARY	DECIMAL(11,4)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Column Name:  Data Type:

Collation:  Default:

Comments:

☐ Primary Key
 ☐ Not Null
 ☐ Unique  
☐ Binary
 ☐ Unsigned
 ☐ Zero Fill  
☐ Auto Increment

Columns | Indexes | Foreign Keys | Triggers | Partitioning | Options

```
CREATE TABLE `employee` (
  `EMP_ID` int(11) NOT NULL AUTO_INCREMENT,
  `EMP_NAME` varchar(45) DEFAULT NULL,
  `EMP_HIRE_DATE` datetime DEFAULT NULL,
  `EMP_SALARY` decimal(11,4) DEFAULT NULL,
  PRIMARY KEY (`EMP_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

Employee Table contains one Primary Key with Auto Increment value.

## b. Domain Classes

We have also one domain class that would be persisting into our database Employee table.

```
package com.journaldev.hibernate.jpa.data;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
public class Employee {
```

```

@Id
@Column(name="EMP_ID")
private long employeeId;
@Column(name="EMP_NAME")
private String employeeName;
@Column(name="EMP_HIRE_DATE")
@Temporal(TemporalType.TIMESTAMP)
private Date employeeHireDate;
@Column(name="EMP_SALARY")
private double employeeSalary;

public long getEmployeeId() {
    return employeeId;
}

public void setEmployeeId(long employeeId) {
    this.employeeId = employeeId;
}

public String getEmployeeName() {
    return employeeName;
}

public void setEmployeeName(String employeeName) {
    this.employeeName = employeeName;
}

public Date getEmployeeHireDate() {
    return employeeHireDate;
}

public void setEmployeeHireDate(Date employeeHireDate) {
    this.employeeHireDate = employeeHireDate;
}

public double getEmployeeSalary() {
    return employeeSalary;
}

public void setEmployeeSalary(double employeeSalary) {
    this.employeeSalary = employeeSalary;
}
}

```

- JPA provides @Entity which will be used for indicating Employee as a persistent domain class. Default mapping would be happening in order to map this persistent entity with its Employee Table. In case you've provided Table name or class name that aren't identical, @Table must be used.



- @Id annotation used for indicating identity of a given Employee instance. Because of discrepancies between attribute name and column name, @column must be provided.
- @Column name annotation takes a parameter of mapped column name.

## 2. Persistence Unit

As we've mentioned earlier, JPA provides an alternative way for bootstrapping JPA framework, it's a persistence.xml file. The minimum amount of this file should look like:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <!-- Will be referenced in Spring Context File -->
  <persistence-unit name="jpa-persistence" transaction-
type="RESOURCE_LOCAL">
    <class>com.journaldev.hibernate.jpa.data.Employee</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/journaldev" />
      <property name="javax.persistence.jdbc.user" value="pankaj"
/>
      <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
      <property name="javax.persistence.jdbc.password"
value="pankaj123" />
    </properties>
  </persistence-unit>
</persistence>
```

Persistence unit should define:

- Persistence unit name. That name will be referenced by Spring context.
- Transaction type – JPA implementation have the choice of managing the resource by itself (RESOURCE\_LOCAL) or having them managed by the application server's JTA implementation.
- Information about database connection.

# 3. Maven Dependencies

All required libraries are listed within pom.xml file that's read by Maven itself.

```
<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.journaldev</groupId>
  <artifactId>Primefaces-Hibernate-JPA-Spring-Integration-
Sample</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>Primefaces-Hibernate-JPA-Spring-Integration-Sample Maven
Webapp</name>
  <url>http://maven.apache.org</url>
  <repositories>
    <repository>
      <id>prime-repo</id>
      <name>PrimeFaces Maven Repository</name>
      <url>http://repository.primefaces.org</url>
      <layout>default</layout>
    </repository>
  </repositories>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <!-- Servlet -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>
    <!-- Faces Implementation -->
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-impl</artifactId>
      <version>2.2.4</version>
    </dependency>
    <!-- Faces Library -->
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-api</artifactId>
```

```

        <version>2.2.4</version>
    </dependency>
    <!-- Primefaces Version 5 -->
    <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>5.0</version>
    </dependency>
    <!-- JSP Library -->
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>2.3.1</version>
    </dependency>
    <!-- JSTL Library -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.1.2</version>
    </dependency>
    <!-- Hibernate 4.3.6 core library library -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.3.6.Final</version>
    </dependency>
    <!-- Hibernate 4.3.6 JPA support -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>4.3.6.Final</version>
    </dependency>
    <!-- MySQL driver connector library -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.31</version>
    </dependency>
    <!-- Spring ORM -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>4.0.3.RELEASE</version>
    </dependency>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>4.0.3.RELEASE</version>
    </dependency>
    <!-- Dependencies for Eclipse JPA Persistence API -->
    <dependency>
        <groupId>org.eclipse.persistence</groupId>

```

```

        <artifactId>eclipselink</artifactId>
        <version>2.5.0-RC1</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

## 4. Hibernate/JPA Spring Configuration

Persisting using of JPA requires an instance of EntityManager. This instance can be acquired by configuring a proper Spring context.

```

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

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.2.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">
    <!-- Enable Spring Annotation Configuration -->
    <context:annotation-config />
    <!-- Scan for all of Spring components such as Spring Service -->
    <context:component-scan base-
package="com.journaldev.spring.service"></context:component-scan>

```

```

    <!-- Necessary to get the entity manager injected into the factory
    bean -->
    <bean
class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPos
tProcessor" />

    <!-- Define Hibernate JPA Vendor Adapter -->
    <bean id="jpaVendorAdapter"

class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="databasePlatform"
            value="org.hibernate.dialect.MySQLDialect" />
    </bean>

    <!-- Entity Manager Factory -->
    <bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
        <property name="persistenceUnitName"
value="hibernate.jpa"></property>
        <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
    </bean>

    <!-- Transaction Manager -->
    <bean id="transactionManager"

class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory"
ref="entityManagerFactory" />
    </bean>

    <!-- Detect @Transactional -->
    <tx:annotation-driven transaction-manager="transactionManager" />
</beans>

```

- JPA require an entityManagerFactory object which is an instance of org.springframework.orm.jpa.LocalEntityFactoryBean. This instance must be provided with the name of persistenceUnit and a JPAVendorAdapter.
- To use @Trasnactional annotation properly, TransactionManager should be defined.
- Default name and location for Spring context configuration is applicationContext.xml and beneath of WEB-INF folder.

## 5. EclipseLink/JPA Spring Configuration

Same configuration would be used for EclipseLink, a small change is required is to provide EclipseLink's JPA vendor. Just change the **jpaVendorAdapter** bean to below and the JPA implementation used will be EclipseLink.

```
<!-- Define EclipseLink JPA Vendor Adapter -->
<bean id="jpaVendorAdapter"

class="org.springframework.orm.jpa.vendor.EclipseLinkJpaVendorAdapter">
    <property name="databasePlatform"
        value="org.eclipse.persistence.platform.database.MySQLPlatform"
    />
    <property name="generateDdl" value="false" />
    <property name="showSql" value="true" />
</bean>
```

## 6. Primefaces Deployment Descriptor

Proper configuration of Spring requires adding of Spring listener into Primefaces' deployment descriptor web.xml application.

```
<?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" metadata-complete="true">
    <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>*.xhtml</url-pattern>
```

```

        </servlet-mapping>
        <context-param>
            <description>State saving method: 'client' or 'server'
            (=default). See JSF Specification 2.5.2</description>
            <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
            <param-value>client</param-value>
        </context-param>
        <listener>
            <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
            </listener>
            <listener>
                <listener-
class>com.sun.faces.config.ConfigureListener</listener-class>
            </listener>
        </web-app>

```

## 7. Spring EmployeeService

Spring service is the interaction point between presentation layer and persistence layer. If you're familiar with DAO, you can consider it something similar.

```

package com.journaldev.spring.service;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import com.journaldev.hibernate.jpa.data.Employee;

@Component
public class EmployeeService {
    @PersistenceContext
    private EntityManager em;

    public EntityManager getEm() {
        return em;
    }

    public void setEm(EntityManager em) {
        this.em = em;
    }

    @Transactional

```

```

    public void register(Employee emp) {
        // Save employee
        this.em.persist(emp);
    }
}

```

EntityManager is injected using @PersistenceContext annotation. Even you've defined an instance of EntityManagerFactory, but a JPA implementation will be very smart to inject you an instance of EntityManager. EntityManager would be something similar for Session in Hibernate. In case you've invoked any of its CRUD operation within both of context and active transaction, your operation would be persisted against your persistence store. Note that em.persist() and using of @Transactional annotation upon register method.

## 8. Primefaces Managed Bean – RegisterEmployee

RegisterEmployee is a faces managed bean that's used for handling user interaction and validation of user's input.

```

package com.journaldev.prime.faces.beans;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;

import com.journaldev.jpa.data.Employee;
import com.journaldev.spring.service.EmployeeService;

@ManagedBean
@SessionScoped
public class RegisterEmployee {

    @ManagedProperty("#{employeeService}")
    private EmployeeService employeeService;

    private Employee employee = new Employee();

    public EmployeeService getEmployeeService() {
        return employeeService;
    }
}

```



```

    public void setEmployeeService(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    public Employee getEmployee() {
        return employee;
    }

    public void setEmployee(Employee employee) {
        this.employee = employee;
    }

    public String register() {
        // Calling Business Service
        employeeService.register(employee);
        // Add message
        FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage("The Employee
"+this.employee.getEmployeeName()+" Is Registered Successfully"));
        return "";
    }
}

```

- Spring service EmployeeService is injected using Spring el-resolver that get declared with your faces-config.xml.
- Register method would delegate the invocation into an injected EmployeeService instance. As such, EmployeeService would handle real registration.

## 9. Primefaces Employee Registration

```

package com.journaldev.prime.faces.beans;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;

import com.journaldev.jpa.data.Employee;
import com.journaldev.spring.service.EmployeeService;

@ManagedBean
@SessionScoped
public class RegisterEmployee {

    @ManagedProperty("#{employeeService}")
    private EmployeeService employeeService;
}

```

```

private Employee employee = new Employee();

public EmployeeService getEmployeeService() {
    return employeeService;
}

public void setEmployeeService(EmployeeService employeeService) {
    this.employeeService = employeeService;
}

public Employee getEmployee() {
    return employee;
}

public void setEmployee(Employee employee) {
    this.employee = employee;
}

public String register() {
    // Calling Business Service
    employeeService.register(employee);
    // Add message
    FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage("The Employee
"+this.employee.getEmployeeName()+" Is Registered Successfully"));
    return "";
}
}

```

EMP_ID	EMP_NAME	EMP_HIRE_DATE	EMP_SALARY
33	Mohammad Amr	2014-08-25 00:00:00	2500.6500
NULL	NULL	NULL	NULL

This tutorial aimed to help you get both of Hibernate and EclipseLink JPA implementations used into your project. JPA has changed your life, it's so

easy to configure, use and track. It's plugged in with a default logging mechanism that would help you find your problem shortly. Contribute us by commenting below and find downloaded source code.

**You can download the source code of “PrimeFaces Spring JPAHibernate EclipseLink Project” from [here](#).**

# Copyright Notice

Copyright © 2015 by Pankaj Kumar, [www.journaldev.com](http://www.journaldev.com)

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law. For permission requests, write to the publisher, addressed “Attention: Permissions Coordinator,” at the email address [Pankaj.0323@gmail.com](mailto:Pankaj.0323@gmail.com).

Although the author and publisher have made every effort to ensure that the information in this book was correct at press time, the author and publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause. Please report any errors by sending an email to [Pankaj.0323@gmail.com](mailto:Pankaj.0323@gmail.com)

All trademarks and registered trademarks appearing in this eBook are the property of their respective owners.

# References

1. <http://www.journaldev.com/2990/primefaces-5-jsf-2-beginners-example-tutorial>
2. <http://www.journaldev.com/3086/primefaces-5-accordionpanel-component-example-tutorial>
3. <http://www.journaldev.com/4096/primefaces-spring-hibernate-integration-example-tutorial>
4. <http://www.journaldev.com/4058/primefaces-spring-4-with-jpa-hibernate-eclipselink-example-tutorial>