

Java Jazz up

A BETTER WAY TO LEARN PROGRAMMING



JBoss

EJB 3.0

JDK-Annotations

JSF

ANT

Struts2

Readers Forums

Tips & Tricks

Discussion Problems

India's Cheapest web Service Provider

 **RoseIndia**

OUR SERVICES

- **ERP Soluiotns**
- **Software Solutions**
- **Web Development**
- **Web Designing**
- **Web Redesigning**
- **Domain Registration**
- **Web Promotion**
- **SEO**
- **Article Writing**
- **Blog Writing**
- **News Writing**
- **SEO Copywriting**
- **Technical Documentation**
- **E-Commerce Solutions**
- **CRM**
- **Outsourcing**

Extend your reach
with our solutions...

<http://www.roseindia.net/services>

"Learning new technologies demands individuals to work together to construct shared understandings and knowledge."

Published by

RoseIndia

JavaJazzUp Team

Editor-in-Chief

Deepak Kumar

Editor-Technical

Ravi Kant

Sr. Graphics Designer

Suman Saurabh

Graphics Designer

Santosh Kumar
Amardeep Patel

**Register with JavaJazzUp
and grab your monthly issue**

"Free"

Editorial

Dear Readers,

We are back here with the **Christmas cum New Year (Jan' 2008)** issue of Java Jazz-up. The current edition is specially designed for the sprouting technocrats. This issue highlights the interesting Java technologies especially for the beginners.

Though it was a hard job to simplify the complexities of the technologies like JBoss AS, Hibernate 3.0, Ant tool, struts 2, JSF and Design Patterns. Still our team has done a marvelous work in making it easy and simpler for the new programmers regime. This issue reflects our consistent attempts to avail the quality technological updates that enforce the readers to appreciate it a lot and be a part of its Readers Community.

Java News and Updates section provides the latest updates of the things happening around the globe making the readers aware of the java technological advancement. In this section, you will know the new features introduced in the existing tools, utilities, application servers, IDEs, along with the Java API updates.

The set of articles conferring technologies like Design patterns, JSF, Hibernate 3.0, Integrating various technologies like JSF, Spring, Hibernate together etc. are provided in such a manner that even a novice learns and implements the concepts in a very easy manner.

We are providing it in a PDF format so that you can view and even download it as a whole and get its hard copy.

Please send us your feedback about this issue and participate in the Reader's Forum with your problems, issues concerned with the topics you want us to include in our next issues.

Editor-in-Chief

Deepak Kumar
Java Jazz up

Content

- 05** [Java News | Apple](#) launched a stack of patches recently fixing at least 18 security vulnerabilities in its implementation of **Java** for **Mac** users. This Java update targets to Mac systems running **OS X 10.4** (Tiger) and earlier versions. Apple claims that none of the vulnerabilities patched in the Java roll-up are there now in **OS X 10.5** (Leopard).
- 07** [Java Developers Desk: Annotations](#) | Sun Microsystems added the features like annotation to make the development easier and more efficient in jdk 5. The main objective to develop the annotations is to make the development easier
- 13** [JBoss Application Server](#) | JBoss is a free, open source application server under the LGPL license that is widely used for developing and deploying enterprise Java applications (J2EE), Web applications, and Portals.
- 15** [EJB 3.0](#) | Enterprise beans are the Java EE server side components that run inside the ejb container and encapsulate the business logic of an enterprise application. Enterprise applications are the software applications developed intended to use at large scale.
- 23** [XML and JAXP](#) | "XML is a cross-platform, software and hardware independent tool for transmitting information"
- 36** [Hibernate with Annotation](#) | The Java 5 version has introduced a powerful way to provide the metadata to the JVM. The mechanism is known as Annotations.
- 39** [Introduction to Ant](#) | Ant is a platform-independent build tool that specially supports for the Java programming language.
- 42** [Struts2 Data Tags](#) | Apache Struts is an open-source framework used to develop Java web applications. We started introducing struts generic tags in the November issue. In this section, we will continue further with the data tags (generic tags) provided with struts 2 framework and the rest will be included in the subsequent issues of the magazine.
- 51** [Integrating JSF, Spring and Hibernate](#) | This article explains integrating JSF (MyFaces), Spring and Hibernate to build real world User Login and Registration Application using MySQL as database. This application lets the new user create an account and existing user access the application by user name and password.
- 77** [Facelet](#) | Facelet is a view technology for Java Server Faces (JSF) that allows building composite views more quickly and easily than with JSP which is the default view technology for JSF. JSP pages are compiled into servlets but it's not the case with Facelets because Facelet pages are XML compliant and its framework uses a fast SAXbased compiler to build views
- 90** [Design Patterns](#) | Behavioral Patterns Behavioral patterns are those patterns, which are specifically concerned with communication (interaction) between the objects. The interactions between the objects should be such that they are talking to each other and are still loosely coupled.
- 96** [Tips & Tricks](#) | Copy content from one file to another This example explains how to copy contents from one file to another file. Copy file is one of the good use of io package of Java. The logic of program is explained below:
- 104** [Advertise with Us](#) | We are the top most providers of technology stuffs to the java community.
- 105** [Valued JavaJazzup Readers Community](#) | We invite you to post Java-technology oriented stuff. It would be our pleasure to give space to your posts in JavaJazzup.

Java News and Releases

I. Apple Patches Java, OS X and Safari 3 Flaws

Apple launched a stack of patches recently fixing at least 18 security vulnerabilities in its implementation of **Java** for **Mac** users. This Java update targets to Mac systems running **OS X 10.4** (Tiger) and earlier versions. Apple claims that none of the vulnerabilities patched in the Java roll-up are there now in **OS X 10.5** (Leopard). However, a fair number of the fixes in the patch batch for OS X also apply to Leopard.

II. Secunia aims to be leading vulnerability intelligence provider

It is Secunia's ambition to be the leading vulnerability intelligence provider and distributor in the world - second to none.

They have detected few of the vulnerability in the Sun Java System Web Proxy Server, These vulnerabilities can be exploited by malicious people to conduct cross-site scripting attacks.

A such kind of vulnerability is reported in 4.x versions prior to 4.0.6. It is being illustrated through an example: Input passed via unspecified parameters within the View Error Log functionality is not properly sanitized before being returned to the user. This can be exploited to execute arbitrary HTML and script code in a user's browser session in context of an affected site.

Solution is just to update Sun Java System Web Proxy Server 4.x to version 4.0.6.

Another vulnerability is reported in 4.x versions prior to 4.0.6 and 3.x versions prior to 3.6 SP11. It is illustrated with an example:

Input passed via unspecified parameters within the View URL Database functionality is not properly sanitised before being returned to the user. This can be exploited to execute arbitrary HTML and script code in a user's browser session in context of an affected site.

Solution is just to update Sun Java System Web Proxy Server 3.x to version 3.6 Service Pack 11.

III. JetBrains added Ruby, Groovy to Java IDE

Now the plug-ins for dynamic languages are available. Recently JetBrains has unveiled the plug-ins that allows the users of its IntelliJ Idea IDE to accommodate Groovy and Ruby programming.

The IDE highly supports java development, is on the way of expansion to accommodate newly popular dynamic languages. To meet the purpose, JetBrains is offering its JetGroovy Plugin 1.0 for Groovy and Grails framework developers. It is also available for Ruby development as the Ruby Plugin 1.0.

IV. Spring Integration: a central service and message bus

Recently, SpringSource announced the creation of Spring Integration, a project aimed to provide a central service and message bus inside the Spring Framework. This is built on the Spring's already-impressive capabilities for providing simple models for using services. Spring Integration is a logical next step for Spring, as

India's Cheapest
web Service
Provider



RoseIndia
Technologies Pvt. Ltd.

Extend your reach

with our solutions...

Web: <http://www.roseindia.net/services/> E-mail: deepak@roseindia.net

Java news and releases

it already provides services for JMS, remoting, scheduling, email, lifecycle management, transaction management, event publication and subscription, and transaction management.

The benefit is that a Spring configuration can manage all of the communication protocol, such that the service barely has to know it's a service for an ESB at all.

V. Major update to WebSphere XD DataGrid shipped

Recently IBM WebSphere XD DataGrid has shipped a major update to the ObjectGrid function known as iFix 3. It includes the ObjectGrid data grid middleware. ObjectGrid provides an embeddable distributed memory platform for implementing complex event processing, network attached memory, HTTP Session management, next generation scalable OLTP databases or XTP style applications. It provides Map based APIs or an EntityManager style API as well as a stored procedure like API. It's very lightweight and scales from a pair of JVMs to thousands distributed across multiple data centers.

VI. Trace Modeler v1.0 released

Trace Modeler v1.0 has been released. It is an easy-to-use and smart UML sequence diagram editor that provides immediate and automatic layout of UML sequence diagrams and simple drag and drop interface. It's cross-platform i.e. runs on any platform. Its main benefit is that it can save much amount of time. It instantly updates a diagram's layout whenever it changes,

freeing you to focus on the actual content. Furthermore, its layout engine ensures that every diagram is visually pleasing and structurally correct. Trace Modeler also offers a couple of unique features like inlining message calls, splitting activations, smart flow-based comment connectors, right-to-left diagram layout and full bidi-text support for non-Western scripts, control flow highlighting, automatic object lifetimes.

VII. Apache ActiveMQ 5.0 released

Apache, powerful and flexible httpd server, comes with an unrestrictive license. Apache ActiveMQ is the most powerful open source Message Broker and Enterprise Integration Patterns provider. It has recently released ActiveMQ 5.0 with lot of new features like AMQ message store, message cursors, blob messages, a command agent, enterprise integration patterns, logging a warning, message transformation and mirrored queues.

VIII. JSFUnit 1.0 Beta 1 released

JSFUnit 1.0, JSF Testing Tool, has been released recently by Jboss. This Beta release allows testing JSF applications based on cactus and Junit. It allows complete integration testing and unit testing of JSF applications. This version will use three different testing tools In-container Testing Framework, Framework for JSF Static Analysis Testing, JSFTimer for Performance Testing of the JSF Lifecycle. JSFUnit allows testing of a running JSF application and even looking at the HTML output of each client request.

Java Developers Desk: Annotations

An Introduction to Annotations

Sun Microsystems added the features like annotation to make the development easier and more efficient in JDK 5. The main objective to develop the annotations is to make the development easier. Annotations behave like the meta. The literal meaning of meta data is data about data. Java also signifies this meaning. Annotations are like meta data, means you are free to add your code and can also apply them to variables, parameters, fields type declarations, methods and constructors. Metadata is also used to create the documentation to perform rudimentary compile time checking and even for tracking down the dependencies in code. XDoclet contains all these features and is widely used. Annotations provide a means of indicating about methods, classes, dependencies, incompleteness and also about the references on other methods and classes respectively. Quoting from Sun's official site, "It (annotation-based development) lets us avoid writing boilerplate code under many circumstances by enabling tools to generate it from annotations in the source code. This leads to a declarative programming style where the programmer says what should be done and tools emit the code to do it."

Annotation is the way of associating the program elements with the meta tags so that the compiler can extract program behavior to support the annotated elements to generate interdependent code when necessary.

Fundamentals of annotations

While going through the annotations you should consider two things. The first one is the "annotation" itself and second one is the "annotation types". An annotation is the meta tag, used to give some life to the code you are using. While annotation type is used to define annotations so that you can use them while creating your own custom annotations.

An annotation type definition appends an "at" @ sign at the start of the interface keyword with the annotation name. On the other hand, an annotation includes the "at" @ sign followed by the annotation type. You can also add the

data within the parenthesis after the annotation name. Let's illustrate the concept more clearly by using some examples.

Defining an annotation (Annotation type)

```
public @interface Example {  
    String showSomething();  
}
```

Annotating the code (Annotation)

```
Example (showSomething="Hi! How r you")  
public void anymethod() {  
    ....  
}
```

Annotation Types:

Three types of annotations types are there in Java.

I. Marker:

Like the marker interface, marker annotations does not contain any elements except the name itself. The example given below clarifies the concept of marker interface.

Example:

```
public @interface Example{  
    }  
    Usage:  
    @Example  
    public void anymethod() {  
        -----  
    }  
}
```

II. Single-value:

This type of elements provide only single value. It means that these can be represented with the data and value pair or we can use the shortcut syntax (just by using the value only within the parenthesis).

Example:

```
public @interface Example{  
    String showSomething();  
}
```

Java Developers Desk: Annotations

Usage:

```
@Example ("Hi ! How r you")
public void anymethod(){
    -----
}
```

Multi-value or Full-value

These types of annotations can have multiple data members. Therefore use full value annotations to pass the values to all the data members.

Example:

```
public @interface Example{
    String showSomething();
    int num;
    String name;
}
```

Usage:

```
@Example (showSomething = "Hi! How r
you", num=5, name="amit" )
    public void anymethod{
        // code here
    }
```

Rules defining the Annotation type:

Here are some rules that one should follow while defining and using annotations types

- Start the annotation declaration starting with the symbol "at" @ following the interface keyword that should follow the annotation name.
- Method declaration should not throw any exception.
- Method declaration should not contain any parameter.
- Method using annotations should return a value, one of the types given below:
- String
- primitive
- enum
- Class
- array of the above types

Categorizing Annotations:

JDK 5 contains two categories of annotations:

Simple annotations:

These types of annotations are used to annotate the code only. We can not use these types of annotations for creating the custom annotation type.

Meta annotations:

Also known as annotations of annotations are used to annotate the annotation-type declaration.

I. Simple annotations:

JDK 5 includes three types of simple annotations.

- Override
- Deprecated
- Suppresswarning

JDK 5 does not include many built-in annotations but it facilitates to core java to support annotation features. Now will discuss in brief each of the above simple annotation types along with examples.

1) Override annotation:

The override annotation ensures that the annotated method is used to override the method in the super class. If the method containing this type of annotation does not override the method in the super class then the compiler will generate a compile time error.

Lets take an example and demonstrate what will happen if the annotated method does not override the method in the super class.

Example 1:

```
public class Override_method{
    @Override
    public String toString(){
        return super.toString() +
            "Will generate an compile time error.";
    }
}
```

Java Developers Desk: Annotations

Suppose there is spell mistake in the method name such as the name is changed from toString to toStrimg. Then on compiling the code will generate the message like this:

```
Compiling 1 source file to D:\tempNew Folder
(2)
TestJavaApplication1\buildclasses
D:\tempNew Folder
(2)TestJavaApplication1\src\test
myannotationTest_Override.java:24: method
does not override a method from its
superclass
@Override
1 error
BUILD FAILED (total time: 0 seconds)
```

2) Deprecated annotation:

These types of annotations ensure that the compiler warns you when you use the deprecated element of the program. The example given below illustrates this concept.

Example: Lets first create the class containing the deprecated method.

```
public class Deprecated_method{
    @Deprecated
    public void showSomething() {
        System.out.println("Method has been
        deprecated");
    }
}
```

Now lets try to invoke this method from inside the other class:

```
public class Test_Deprication {
    public static void main(String arg[]) throws
    Exception {
        new Test_Deprication();
    }
    public Test_Deprication() {
        Deprecated_method d = new
        Deprecated_method();
        d.showSomething();
    }
}
```

The method showSomething() in the above example is declared as the deprecated method. That means we can't further use this

method any more. On compiling the class Deprecated_method does not generate any error. While compiling the class Test_Deprication generates the message like this:

```
Compiling 1 source file to D:\tempNew Folder
(2)TestJavaApplication1\buildclasses
D:\tempNew Folder
(2)TestJavaApplication1\src\test\myannotation
Test_Deprication.java:27:
warning: [deprecation] showSomething() in
test.myannotation.Deprecated_method has
been deprecated
d.showSomething();
1 warning
```

3) Suppresswarning annotation:

These types of annotations ensure that the compiler will shield the warning message in the annotated elements and also in all of its sub-elements. Lets take an example:

Suppose you annotate a class to suppress a warning and one of its method to suppress another warning, then both the warning will be suppressed at the method level only. Lets demonstrate it by an example:

```
public class Test_Depricated {
    public static void main(String arg[]) throws
    Exception {
        new TestDepricated().showSomething();
    }
    @SuppressWarnings({"deprecation"})
    public void showSomething() {
        Deprication_method d = new
        Deprication_method();
        d.showSomething();
    }
}
```

This example is suppressing the deprecation warnings that means we can't see the warnings any more.

Note: Applying annotation at most deeply nested elements is a good idea. It is better to apply annotations at the method level rather than the class to annotate a particular method.

Java Developers Desk: Annotations

II. Meta-Annotations (Annotation Types):

There are four types of Meta annotations (or annotations of annotations) defined by the JDK 5. These are as follows:

- Target
- Retention
- Documented
- Inherited

1) Target annotation:

Target annotation specifies the elements of a class to which annotation is to be applied. Here is the listing of the elements of the enumerated types as its value:

- @Target(ElementType.TYPE)— applicable to any element of a class.
- @Target(ElementType.FIELD)—applicable to field or property.
- @Target(ElementType.PARAMETER)— applicable to the parameters of a method.
- @Target(ElementType.LOCAL_VARIABLE)— applicable to local variables.
- @Target(ElementType.METHOD)— applicable to method level annotation.
- @Target(ElementType.CONSTRUCTOR)— applicable to constructors.
- @Target(ElementType.ANNOTATION_TYPE)— specifies that the declared type itself is an annotation type.

Here is an example that demonstrates the target annotation:

Example:

```
@Target(ElementType.METHOD)
public @interface Test_Element {
    public String doTestElement();
}
```

Now lets create a class that use the Test_Element annotation:

```
public class Test_Annotations {
    public static void main(String arg[]) {
        new Test_Annotations().doTestElement();
    }
    @Test_Target(doTestElement="Hi ! How r you")
    public void doTestElement() {
        System.out.printf("Testing Target Element annotation");
    }
}
```

The **@Target(ElementType.METHOD)** specifies that this type of annotation can be applied only at method level. Compiling and running the above program will work properly. Lets try to apply this type of annotation to annotate an element:

```
public class Test_Annotations {
    @Test_Target(doTestElement="Hi! How r you")
    private String str;
    public static void main(String arg[]) {
        new Test_Annotations().doTestElement();
    }
    public void doTestElement() {
        System.out.printf("Testing Target Element annotation");
    }
}
```

Here we are trying to apply

@Target(ElementType.METHOD) at the field level by declaring the element **private String str;** after the

@Test_Target(doTestElement="Hi ! How r you") statement.

On compiling this code will generate an error like this:

```
"Test_Annotations.java":
D:\R_AND_D\Test_Annotationsrctestmyannotation
Test_Annotations.java:16:
annotation type not applicable to this kind of
declaration at line
16, column 0
```

Java Developers Desk: Annotations

```
@Test_Target(doTestElement="Hi ! How r  
you")  
^  
Error in javac compilation
```

2) Retention annotation:

These types of annotation specify where and how long annotation with this types are to be retained. There are three type of Retention annotations are of three types.

- **RetentionPolicy.SOURCE:** This type of annotation will be retained only at source level and the compiler will ignore them.
- **RetentionPolicy.CLASS:** This type of annotation will be retained at the compile time the virtual machine (VM) will ignore them.
- **RetentionPolicy.RUNTIME:** Virtual machine will retained the annotation of this type and they can be read only at run-time.
- Lets demonstrate that how this type of annotations are applied by taking an example using RetentionPolicy.RUNTIME.

Example:

```
@Retention(RetentionPolicy.RUNTIME)  
public @interface Retention_Demo {  
String doRetentionDemo();  
}
```

This example uses the annotation type @Retention(RetentionPolicy.RUNTIME) that indicates the VM will retained your Retention_Demo annotation so that it can be read effectively at run-time.

3) Documented annotation:

This type of annotation should be documented by the javadoc tool. javadoc does not include the annotation by default. Include the

annotation type information by using **@Documented** in the generated document. In this type of annotation all the processing is done by javadoc-like tool.

The given example demonstrates the use of the **@Documented** annotations.

Example:

```
@Documented  
public @interface Documented_Demo {  
String doTestDocumentedDemo();  
}
```

Next, make changes in Test_Annotations class as follows:

```
public class Test_Annotations {  
public static void main(String arg[]) {  
new  
Test_Annotations().doTestRetentionDemo();  
new  
Test_Annotations().doTestDocumentedDemo();  
}  
@Retention_Demo  
(doTestRetentionDemo="Hello retention  
annotation")  
public void doTestRetentionDemo() {  
System.out.printf("Testing 'Retention'  
annotation");  
}  
@Documented_Demo  
(doTestDocumentedDemo="Hello Test  
documentation")  
public void doTestDocumentedDemo() {  
System.out.printf("Testing 'Documented'  
annotation");  
}  
}
```

4) Inherited Annotation:

This annotation is little bit complex. It inherits the annotated class automatically. If you specify **@Inherited** tag before defining a class then apply the annotation at your class and finally extend the class then the child class inherits the properties of the parent class automatically. Lets demonstrate the benefits of using the

Java Developers Desk: Annotations

@Inherited tag by an example:

Example:

Lets first, define the annotation:

```
@Inherited
public @interface ParentObjectDemo {
    boolean isInherited() default true;
    String showSomething() default "Show
anything?";
}
```

Have you seen the difference? You have to implement all the methods of the parent interface. You will have to implement the equals(), toString(), and the hashCode() methods of the Object class and also the annotation type method of the java.lang.annotation.Annotation class. You will also have to include all these methods in your class regardless of whether you are implementing all these methods or not

Now, annotate the class with our annotation:

```
@ParentObjectDemo
public Class ChildObjectDemo {
}
```

The above example shows that you do not need to define the interface methods inside the implemented class. The @Inherited tag automatically inherits the methods for you. Suppose you define the implementing class in the old-fashioned-java-style then let us see the effect of doing this:

```
public class ChildObjectDemo implements
ParentObjectDemo {
    public boolean isInherited() {
        return false;
    }
    public String showSomething() {
        return "";
    }
    public boolean equals(Object obj) {
        return false;
    }
    public int hashCode() {
        return 0;
    }
    public String toString() {
        return "";
    }
    public Class annotationType() {
        return null;
    }
}
```

JBoss Application Server

Introduction to JBoss Application Server

JBoss is a free, open source application server under the LGPL license that is widely used for developing and deploying enterprise Java applications (J2EE), Web applications, and Portals. It provides the full features of J2EE 1.4 such as EJB container as well as extended enterprise services (EJB) including such as database access (JDBC), transactions (JTA), messaging (JTS), naming (JNDI) and management support (JMX). It also provides enterprise-class security, and resource management.

JBoss is a cross-platform Java-based AS, due to this reason it is usable on any operating system that Java supports.

Features of JBoss AS:

JBoss is advanced middleware with a full J2EE based personality.

I. Open Standards and Open Source:

JBoss is an open source J2EE 1.4 certified AS having business friendly license that allows the developers to free download, use, embed, and distribute the JBoss AS.

II. Simplicity:

JBoss AS supports full features of J2EE 1.4 including EJB, JCA, JSP, JMX, HTTP etc. It provides a bridge for the enterprise Java programming model, and enables developers to get started quickly and easily with these applications.

III. Clustering and High Availability:

JBoss AS provides the clustering of any java objects (EJB, HTTP, POJO), load balancing, and distributed deployment features that are required for deploying large scalable enterprise applications.

IV. 100% Pure Java:

JBoss is pure Java-based AS. Due to this reason, it is interoperable with most operating systems that are capable of running a Java Virtual Machine (JVM). These OS includes Red Hat Enterprise Linux, SUSE Linux, Microsoft Windows, Sun Solaris, HP-UX, and others.

V. Supporting for Breed Technologies:

JBoss AS integrates JAAS, Hibernate, Apache Tomcat, EJB 3.0, Aspect Oriented Programming (AOP) and JBoss Cache into its microkernel foundation approach that is based on Java Management eXtensions (JMX).

JBoss AS Versions:

JBoss Application Server has been released in several versions with their sub-versions listed below.

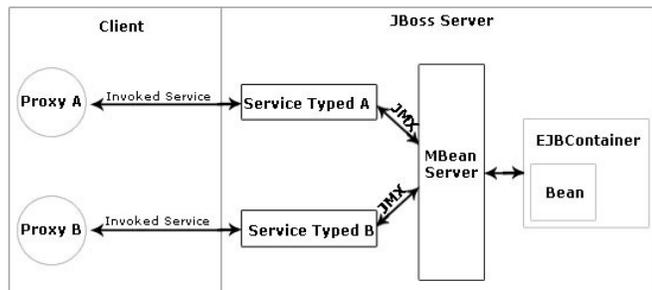
- JBoss Application Server was started as an open source EJB container in 1999.
- JBoss 2.x was a full J2EE 1.2 based server. JBoss 3.x was a J2EE 1.3 based server.
- JBoss 4.x is our current J2EE 1.4 production version.
- JBoss 5.x as beta version is a Java EE 5 certified based server.
- JBoss AS 4.2.x versions support for EJB3.

JBoss 3.x was released to provide full framework for building such applications that are based on a Java microkernel (**JMX**) and service oriented architecture (**SOA**). While **JBoss 4.x** explores aspect oriented middleware (**AOM**) and Java based middleware independent of J2EE.

JBoss AS 5.x Beta versions include the following core technologies such as POJO based microcontainer, EJB 3.0, Hibernate 3.2 - JPA certified, JBoss Messaging, JBoss WebServices 2.0 (JAX-WS), JBoss Seam 1.1, etc.

JBoss AS Architecture:

JBoss Application Server



Working Process of JBoss AS:

The JBoss creates an **MBean server** instance in one of the first steps when it starts up. Then, it plugs the manageable MBean components by registering with the MBean server that is a registry for Mbeans.

JBoss implements the classloading M-Let service dynamically, which treats as an agent service. This service allows registering of the MBeans (specified in a text based configuration files) to the MBean server.

The functionality is provided by MBeans actually instead of the JMX MBean server. The MBean server is only the sense of a microkernel aggregator component that interconnects the Mbeans.

In the architecture of Jboss we can also see the EJB Container as the core implementation of JBoss server that supports its Plugins, InstancePool, EntityPersistenceManager, StatefulSessionPersistenceManager, to provide EJB services to a particular EJB.

Getting familiar with JBoss AS 4.2.1.GA

JBoss AS 4.2.1.GA is the first bug fixing release version of the JBoss Application Server v4.2 series. Its aim is to fix the most important bugs against JBossAS v4.2.0.GA that are reported by the community. There are a few minor components are upgraded (Hibernate, JacORB, JBoss TS, JBoss Remoting and Sun JSF) in this released version.

In this tutorial of JBoss AS, we have used **JBoss AS 4.2.1.GA** version in which you will learn, how to deploy and run an EJB-based application.

I. Downloading and Installing JBoss AS 4.2.1.GA

The JBoss application server 4.2.1.GA is available as a free download from the JBoss website. You can download this version from <http://labs.jboss.com/jbossas/downloads/> URL by clicking the **Download** button or **click here** to extract files to save in your disk.

JBoss 4.2.1.GA Platform is easy to install. It only requires at least a Java 1.4 or Java 1.5 JVM. Apart from this, also make sure for the JAVA_HOME environment variable that is to be set to point the JDK installation.

Once you have installed JBoss 4.2.1.GA, the next step is to learn that how to start the JBoss server.

II. Starting the Server

After installing JBoss AS, you will find a **bin** directory inside the main JBoss directory, which contains various scripts. Click on the **run.bat** file to start the server on Windows, and then see the log messages from all the JBoss components as they are deployed and started up. The last message (obviously with different values for the time and start-up speed) should look like the following.

This message verifies that the JBoss as a web server is running on port 8080 (Make sure you don't have anything else already on your machine using that port).

III. Stopping the Server

To stop the server, you can either press **Ctrl+C** on the console or you can run the shutdown script **shutdown.bat** from the **bin** directory. Alternatively, you can use the management console. Look for type=Server under the jboss.system domain and invoke the shutdown operation.

EJB 3.0

Introduction To Enterprise Java Bean 3.0 (EJB 3.0)

Enterprise beans are the Java EE server side components that run inside the ejb container and encapsulate the business logic of an enterprise application. Enterprise applications are the software applications developed intended to use at large scale. These applications involve large number of data accessing concurrently by many users. Enterprise beans are used to perform various types of task like interacting with the client, maintaining session for the clients retrieving and holding data from the database and communicating with the server.

The Enterprise JavaBeans specification defines an architecture for a transactional, distributed object system based on server-side components. These server-side components are called enterprise beans or distributed objects that are hosted in Enterprise JavaBean containers where it provide remote services for clients distributed throughout the network.

The EJB Container

An EJB container is nothing but the program that runs on the server and implements the EJB specifications. EJB container provides special type of the environment suitable for running the enterprise components. The EJB container manages remote access to the bean, security, persistence, transactions, concurrency, and access to and pooling of resources. It hosts an enterprise bean in the same manner that the Java Web Server hosts a servlet or an HTML. An enterprise bean can't perform functions outside of an EJB container.

Migration from EJB2 to EJB3

Migrating to EJB 3.0 is a big step towards simplifying the process of developing EJBs, which reduces lots of complexities, time and cost. In spite of being rich featured, developers feel complex working with previous versions of EJB.

Migration from EJB 2.1 to EJB 3.0

Lets go through some points justifying reasons to adopt EJB 3.0 instead of EJB 2.1:

1. In EJB 2.1, home interface extends the javax.ejb.EJBHome interface and local home interface extends the javax.ejb.EJBLocalHome interface. The EJB 2.1 remote interface extends the javax.ejb.EJBObject interface and local interface extends the javax.ejb.EJBLocalObject interface. In EJB 3.0, home and component interfaces are replaced with POJI business interfaces.
2. EJB 2.1 needs the developer to implement a variety of callback methods in the bean class, like `ejbActivate()`, `ejbPassivate()`, `ejbLoad()`, and `ejbStore()`, most of which were never used. EJB 3.0 doesn't force to implement any of these methods and instead can designate any arbitrary method as a callback method to receive notifications for life cycle events.
3. In EJB 2.1, session bean implements the `SessionBean` interface and entity bean implements the `EntityBean` interface. In EJB 3.0, session and entity bean classes are POJOs and do not implement the `SessionBean` and `EntityBean` interfaces.
4. The deployment descriptor, which specifies the EJB name, the bean class name, the interfaces, the finder methods etc. is not required because they are replaced by metadata annotations in the bean classes. Annotations are available in JDK 5.0 so you need JDK 5.0 to develop EJB 3.0 EJBs.
5. In EJB 2.1, client application finds a reference to entity and session bean objects using JNDI name but in EJB 3.0, client finds them using dependency annotations like `@Resource`, `@Inject`, and `@EJB`.
6. In EJB 2.1, developers used their own way to perform database specific operations

EJB 3.0

like primary key generation while EJB 3.0 provides support for several database-specific operations. The O/R mapping model has intrinsic support for native SQL. The O/R mapping is specified using annotations.

7. Runtime services like transaction and security are often implemented as the interceptor methods managed by the container. However, in EJB 3.0 developers can write custom interceptor. So developers have control for the actions like committing transaction, security check, etc.

What is new in EJB 3.0?

Now, have a look over the new features of EJB 3.0 that achieved some simplicity over the previous EJB APIs in various ways:

1. EJBs are now Plain Old Java Objects (POJOs)
2. No need of home and object interface.
3. No need of any component interface.
4. Unnecessary artifacts and lifecycle methods are optional
5. Use of java annotations instead of using XML descriptors
6. Use of dependency injection to simplify client view
7. Simplify APIs to make flexible for bean's environment
8. Defaults are assumed whenever possible

Types of EJB

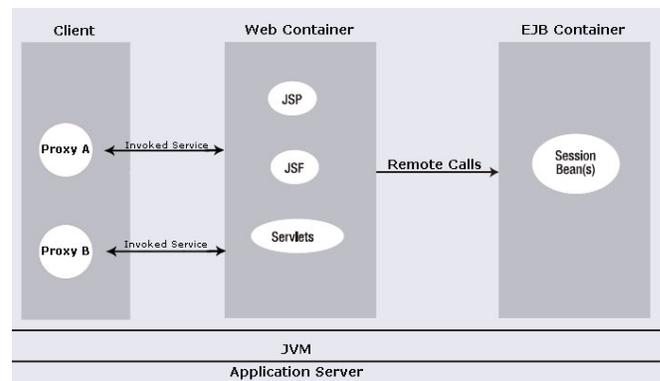
There are three different types of EJB that are suited to different purposes:

- **Session EJB**—A Session EJB is useful for mapping business process flow (or equivalent application concepts). Session

EJBs commonly represent “pure” functionality that is created, as it is needed.

- **Entity EJB**—An Entity EJB maps a combination of data (or equivalent application concept) and associated functionality. Entity EJBs are usually based on an underlying data store and will be created based on that data within it.
- **Message-driven EJB**—A Message-driven EJB is very similar in concept to a Session EJB, but is only activated when an asynchronous message arrives.

Session Bean on EJB Container



Session beans are divided into two parts.

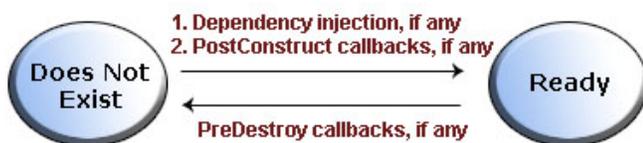
- **Stateless:** A session bean is the enterprise bean that directly interacts with the user and contains the business logic of the enterprise application. A session bean represents a single client accessing the enterprise application deployed on the server by invoking its method. An application may contain multiple sessions depending upon the number of users accessing to the application.
- **Stateful:** These types of beans use the instance variables that allow the data persistent across method invocation because the instance variables allow persistence of data across method

invocation. The client sets the data to these variables, which he wants to persist. Stateful session beans have the extra overhead for the server to maintain the state than the stateless session bean.

- In this tutorial, you will learn how a stateless EJB application is developed using an Application Server Jboss 4.2.0. So let's first see the life cycle of a **Stateless Session Bean**.

Life Cycle of a Stateless Session Bean:

Since the Stateless session bean does not passivate across method calls therefore a stateless session bean includes only two stages. Whether it does not exist or ready for method invocation. A stateless session bean starts its life cycle when the client first obtains the reference of the session bean. For this, the container performs the dependency injection before invoking the annotated **@PreConstruct** method if any exists. After invoking the annotated **@PreConstruct** method the bean will be ready to invoke its method by the client.



The above figure demonstrates how the Stateless Session Beans are created and destroyed.

The container calls the annotated **@PreDestroy** method while ending the life cycle of the session bean. After this, the bean is ready for garbage collection.

In this tutorial, we are going to develop a Stateless Session Bean Application named **example**. The purpose of **example** is to perform the mathematical operations such as Addition, Subtraction, Multiplication, and Division.

The **example** application consists of an enterprise bean, which performs the calculations, and a **web client**.

There are following steps that you have to follow to develop a calculator JEE application.

1. Create the enterprise bean: **CalculatorBean**
2. Create web clients: **index.jsp**, **form.jsp**, **WebClient.jsp**
3. Deploy **example** onto the server.
4. Using a browser, run the web client.

1. Creating the enterprise bean:

The enterprise bean in our example is a stateless session bean called **CalculatorBean**. The source code for **CalculatorBean** is in "com.javajazz/examples/ejb3/stateless" directory.

Creating **CalculatorBean** requires these steps:

- 1) Coding the bean's Remote business interface and Enterprise bean class.
- 2) Compiling the source code with the Ant tool.

(i) Coding the Business Interface

The business interface defines the business methods that a client can call remotely. The business methods are implemented in the enterprise bean class. The source code for the **CalculatorRemote** business interface is given below.

```
package
com.javajazzup.examples.ejb3.stateless;
import java.math.*;
import javax.ejb.Remote;
import java.lang.annotation.*;
@Remote
public interface CalculatorRemote {
public float add(float x, float y);
public float subtract(float x, float y);
public float multiply(float x, float y);
public float division(float x, float y);
}
```

EJB 3.0

Note that, the **@Remote** annotation decorating the interface definition. This lets the container know that remote clients will access **CalculatorBean**.

(ii) Coding the Enterprise Bean Class

The enterprise bean class for this example is called **CalculatorBean**. This class implements the four business methods (add, subtract, multiply, division) that are defined in the **CalculatorRemote** business interface. The source code for the **CalculatorBean** class is given below.

```
package
com.javajazzup.examples.ejb3.stateless;
import java.math.*;
import javax.ejb.Stateless;
import javax.ejb.Remote;
@Stateless(name="CalculatorBean")
@Remote(CalculatorRemote.class)
public class CalculatorBean implements
CalculatorRemote{
    public float add(float x, float y){
        return x + y;
    }
    public float subtract(float x, float y){
        return x - y;
    }
    public float multiply(float x, float y){
        return x * y;
    }
    public float division(float x, float y){
        return x / y;
    }
}
```

Note that, the **@Stateless** annotation decorating the enterprise bean class. This lets the container know that **CalculatorBean** is a stateless session bean.

2. Creating a Web Client

The web client is divided into two pages. First is "**form.jsp**" where a request form is sent to the client; second is "**WebClient.jsp**" which is called from the "form.jsp" page.

A JSP page is a text-based document that contains JSP elements, which construct dynamic content, and static template data, expressed

in any text-based format such as **HTML**, **WML**, and **XML**.

The source code for the "**form.jsp**" is given below.

```
<html>
  <head>
    <title>Calculator</title>
  </head>
  <body bgcolor="pink">
    <h1>Calculator</h1>
    <hr>
    <form action="WebClient.jsp"
method="POST">
      <p>Enter first value:
        <input type="text" name="num1"
size="25"></p>
        <br>
        <p>Enter second value:
          <input type="text" name="num2"
size="25"></p>
          <br>
          <b>Select your choice:</b><br>
          <input type="radio" name="group1" value
="add">Addition<br>
          <input type="radio" name="group1" value
="sub">Subtraction<br>
          <input type="radio" name="group1" value
="multi">Multiplication<br>
          <input type="radio" name="group1" value
="div">Division<br>
          <p>
            <input type="submit"
value="Submit">
            <input type="reset"
value="Reset"></p>
          </form>
        </body>
      </html>
```

The following statements given below in "**WebClient.jsp**" are used for locating the business interface, creating an enterprise bean instance, and invoking a business method.

```
InitialContext ic = new InitialContext();
CalculatorRemote calculator =
(CalculatorRemote)ic.lookup("example/
CalculatorBean/remote");
```

EJB 3.0

The classes needed by the client are declared using a JSP **page** directive (enclosed within the `<%@ %>` characters). Because locating the business interface and creating the enterprise bean are performed only once, this code appears in a JSP **declaration** (enclosed within the `<%! %>` characters) that contains the initialization method, **jspInit**, of the JSP page. A **scriptlet** (enclosed within the `<% %>` characters) retrieves the parameters from the request and converts it to a **Float** object. Finally, a JSP scriptlet invokes the enterprise bean's business methods, and JSP **expressions** (enclosed within the `<%= %>` characters) insert the results into the stream of data returned to the client.

The full source code for the **WebClient.jsp** is given below.

```
<%@ page contentType="text/html;
charset=UTF-8" %>
<%@ page
import="com.javajazzup.examples.ejb3.stateless.*,
javax.naming.*"%>
<%!
    private CalculatorRemote calculator = null;
    float result=0;
    public void jspInit() {
        try {
            InitialContext ic = new
InitialContext();
            calculator = (CalculatorRemote)
ic.lookup("example/CalculatorBean/remote");
            System.out.println("Loaded Calculator
Bean");
        } catch (Exception ex) {
            System.out.println("Error:" +
ex.getMessage());
        }
    }
    public void jspDestroy() {
        calculator = null;
    }
%>
    <%
        try {
            String s1 =
request.getParameter("num1");
            String s2 = request.getParameter("num2");
            String s3 =
request.getParameter("group1");
            if ( s1 != null && s2 != null ) {
                Float num1 = new Float(s1);
```

```
                Float num2 = new Float(s2);
                if(s3.equals("add"))
                    result=calculator.add(num1.floatValue(),
num2.floatValue());
                else if(s3.equals("sub"))
                    result=calculator.subtract(num1.floatValue(),
num2.floatValue());
                else if(s3.equals("multi"))
                    result=calculator.multiply(num1.floatValue(),
num2.floatValue());
                else
                    result=calculator.division(num1.floatValue(),
num2.floatValue());
            }
        }
    }
    <p>
    <b>The result is:</b> <%= result %>
    <p>
    <%
}
} // end of try
catch (Exception e) {
    e.printStackTrace ();
}
%>
```

The source code for the **"index.jsp"** is given below that will actual call the client-design form.

```
<%@page language="java" %>
<html>
<head>
<title>Ejb3 Stateless Tutorial</title>
</head>
<body bgcolor="#FFFFCC">
<p align="center"><font size="6"
color="#800000"><b>Welcome to <br>
Ejb3-Jboss 4.2.0 Tutorial</b></font>
Click <a href="ejb3/form.jsp">Calculator
Example</a> to execute Calculator<br></p>
</body>
</html>
```

3. Deploy calculator application on the Application Server

To deploy the created **example** application we are going to use Jboss 4.2.0 Application Server about which you have read in the previous

EJB 3.0

section of this Javajazzup issue. So you first need to download the following tools to deploy this application.

- **JDK 1.5 or Higher**
- **apache-ant-1.7.0**
- **JBoss 4.2.1**

Do the following steps to deploy the calculator application:

- (i) Make a directory structure. You can **Click here** to extract the readymade directory structure according to this tutorial.
- (ii) Create the essential deployment descriptor **.xml** files.

build.xml

```
<?xml version="1.0"?>
<project name="Jboss Tutorials" default="all"
basedir=".">
  <target name="init">
    <!-- Define -->
    <property name="dirs.base"
value="\${basedir}"/>
    <property name="classdir"
value="\${dirs.base}/build/classes"/>
    <property name="src" value="\${dirs.base}/
src"/>
    <property name="web"
value="\${dirs.base}/web"/>
    <property
name="deploymentdescription"
value="\${dirs.base}/deploymentdescriptors"/
>
    <property name="warFile"
value="example.war"/>
    <property name="earFile"
value="example.ear"/>
    <property name="jarFile"
value="example.jar"/>
    <property name="earDir"
value="\${dirs.base}/build/ear"/>
    <property name="warDir"
value="\${dirs.base}/build/war"/>
    <property name="jarDir"
value="\${dirs.base}/build/jar"/>
    <!-- classpath for Project -->
    <path id="library.classpath">
    <pathelement path="libext/servlet-
```

```
api.jar"/>
    <pathelement path="libext/ejb3-
persistence.jar"/>
    <pathelement path="libext/javaee.jar"/
>
    <pathelement path="\${classpath}"/>
  </path>
  <!-- Create Web-inf and classes
directories -->
  <mkdir dir="\${warDir}/WEB-INF"/>
  <mkdir dir="\${warDir}/WEB-INF/
classes"/>
  <!-- Create Meta-inf and classes
directories -->
  <mkdir dir="\${earDir}/META-INF"/>
  <mkdir dir="\${jarDir}/META-INF"/>
  <mkdir dir="\${classdir}"/>
</target>
<!-- Main target -->
<target name="all"
depends="init,build,buildWar,buildJar,buildEar"/
>
  <!-- Compile Java Files and store in /build/
src directory -->
  <target name="build" >
    <javac srcdir="\${src}"
destdir="\${classdir}" debug="true"
includes="**/*.java" >
    <classpath refid="library.classpath"/>
  </javac>
</target>
<!-- Create the web archive File -->
<target name="buildWar" depends="init">
  <copy todir="\${warDir}/WEB-INF/
classes">
    <fileset dir="\${classdir}"
includes="**/*.class" />
  </copy>
  <copy todir="\${warDir}/WEB-INF">
    <fileset
dir="\${deploymentdescription}/web/"
includes="web.xml" />
  </copy>
  <copy todir="\${warDir}">
    <fileset dir="\${web}" includes="**/
*.*/" />
  </copy>

  <!-- Create war file and place in ear
directory -->
  <jar jarfile="\${earDir}/
\${warFile}" basedir="\${warDir}" />
</target>
```

EJB 3.0

```
<!-- Create the jar File -->
<target name="buildJar" depends="init">
  <copy todir="{jarDir}">
    <fileset dir="{classdir}"
includes="**/*.class" />
  </copy>
  <copy todir="{jarDir}/META-INF">
    <fileset
dir="{deploymentdescription}/jar/"
includes="ejb-jar.xml,weblogic-cmp-rdbms-
jar.xml,weblogic-ejb-jar.xml" />
  </copy>
  <!-- Create jar file and place in ear
directory -->
  <jar jarfile="{earDir}/{jarFile}"
basedir="{jarDir}" />
</target>
<!-- Create the ear File -->
<target name="buildEar" depends="init">
  <copy todir="{earDir}/META-INF">
    <fileset
dir="{deploymentdescription}/ear"
includes="application.xml" />
  </copy>
  <!-- Create ear file and place in ear
directory -->
  <jar jarfile="../{earFile}"
basedir="{earDir}" />
  <copy todir="C:/jboss-4.2.0.GA/server/
default/deploy/">
    <fileset dir=".." />
  </copy>
  </target>
</project>
```

Put this file in the base (stateless\code) directory.

application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/
ns/javaee" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" version="5"
xsi:schemaLocation="http://java.sun.com/
xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_5.xsd">
  <display-name>Stateless Session Bean
Example</display-name>
  <module>
  <web>
```

```
<web-uri>example.war</web-uri>
<context-root>/example</context-root>
</web>
</module>
<module>
  <ejb>example.jar</ejb>
</module>
</application>
```

Put this file in the **Stateless\code\deploymentdescriptors\ear** directory.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun
Microsystems, Inc.//DTD Web Application 2.3/
/EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app >
</web-app>
```

Put this file in the **Stateless\code\deploymentdescriptors\web** directory.

Put all **.jsp** files in the **Stateless\code\web** directory.

Put all **.java** files in the **Stateless\code\src** directory.

(iii) Start command prompt, and go to the **Stateless\code** directory. Then type the command as:

C:\Stateless\code>ant build.xml

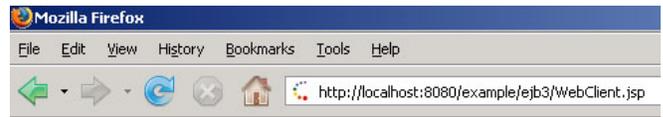
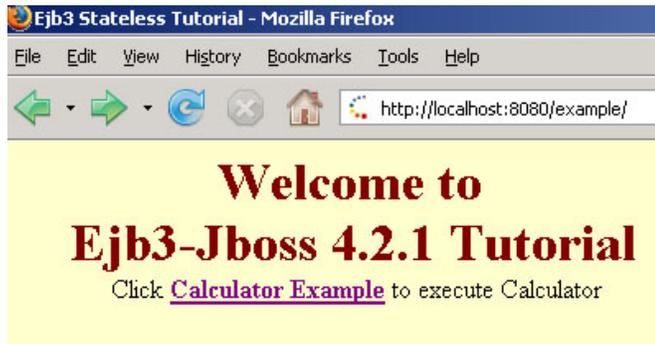
The Ant tool will deploy the **example.ear** file to the **jboss-4.2.0.GA\server\default\deploy** directory.

4. Running the example application Web Client

Open the web browser and type the following URL to run the application:

EJB 3.0

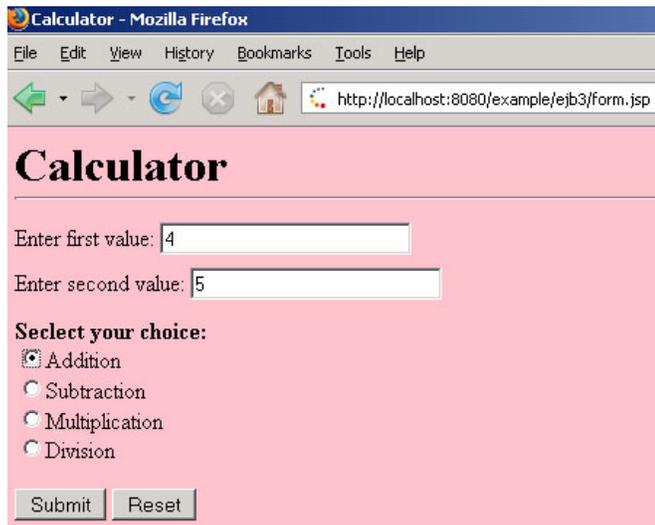
<http://localhost:8080/example>



The result is: 9.0

Download the full source code

Click at the given link as **Calculator Example**:



Give values to the textbox and choose the desire option button as **Addition** then clicks the **Submit** button to get the result.

XML and JAXP

Introduction to XML

About XML

“XML is a cross-platform, software and hardware independent tool for transmitting information”

XML is a W3C Recommendation. It stands for Extensible Markup Language. It is a markup language much like HTML used to describe data. In XML, tags are not predefined. A user defines his own tags and XML document structure like Document Type Definition (DTD), XML Schema to describe the data. Hence it is self-descriptive too. There is nothing special about XML. It is just plain text with the addition of some XML tags enclosed in angle brackets. In a simple text editor, the XML document is easily visible.

Reasons of using XML

There are number of reasons that contribute to the XML's increasing acceptance, few of them are:

1. Plain Text

In XML it is easy to create and edit files with anything from a standard text editor to a visual development environment. XML also provides scalability for anything from small configuration files to a company-wide data repository.

2. Data Identification

The markup tags in XML documents identify the information and break up the data into parts for example.. a search program can look for messages sent to particular people from the rest of the message. Different parts of the information are identified and further they can be used in different ways by different applications.

3. Stylability

When display matters, the style sheet standard, XSL (an advance feature of XML), lets you dictate over the conventional designs (like using HTML) to portray the data. XML being style-free, uses different style sheets to produce

output in postscript, TEX, PDF, or some new format that hasn't even been invented yet. A user can use a simple XML document to display data in diverse formats like

- A plain text file
- An XHTML file
- A WML (Wireless Markup Language) document suitable for display on a PDA
- An Adobe PDF document suitable for hard copy
- A VML (Voice Markup Language) dialog for a voicemail information system
- An SVG (Scalable Vector Graphic) document that draws pictures of thermometers and water containers

4. Universally Processed

Apart from being valid, restrictions are imposed on an xml file to abide by a DTD or a Schema to make it well formed. Otherwise, the XML parser won't be able to read the data. XML is a vendor-neutral standard, so a user can choose among several XML parsers to process XML data.

5. Hierarchical Approach

XML documents get benefited from their hierarchical structure. Hierarchical document structures are, faster to access. They are also easier to rearrange, because each piece is delimited. This makes xml files easy to modify and maintain.

6. Inline Reusability

XML documents can be composed of separate entities. XML entities can be included “in line” in a XML document. And this included sections look like a normal part of the document. A user can single-source a section so that an edit to it is reflected everywhere the section is used, and yet a document composed looks like a one-piece document.

XML and JAXP

Applications using XML

Although there are countless applications that use XML, here are a few examples of the applications that are making use of this technology.

Refined search results - With XML-specific tags, search engines can give users more refined search results. A search engine seeks the term in the tags, rather than the entire document, giving the user more precise results.

EDI Transactions - XML has made electronic data interchange (EDI) transactions accessible to a broader set of users. XML allows data to be exchanged, regardless of the computing systems or accounting applications being used.

Cell Phones - XML data is sent to some cell phones, which is then formatted by the specification of the cell phone software designer to display text, images and even play sounds!

File Converters - Many applications have been written to convert existing documents into the XML standard. An example is a PDF to XML converter.

VoiceXML - Converts XML documents into an audio format so that a user can listen to an XML document.

History of XML

In the 1970's, Charles Goldfarb, Ed Mosher and Ray Lorie invented GML at IBM. GML was used to describe a way of marking up technical documents with structural tags. The initials stood for Goldfarb, Mosher and Lorie.

Goldfarb invented the term "mark-up language" to make better use of the initials and it became the Standard Generalised Markup Language.

In 1986, SGML was adopted by the ISO.

SGML is just a specification for defining markup languages.

SGML (Standardized Generalized Markup Language) is the mother of all markup

languages like HTML, XML, XHTML, WML etc...

In 1986, SGML became an international standard for defining the markup languages. It was used to create other languages, including HTML, which is very popular for its use on the web. HTML was made by Tim Berners Lee in 1991.

While on one hand SGML is very effective but complex, on the other, HTML is very easy, but limited to a fixed set of tags. This situation raised the need for a language that was as effective as SGML and at the same time as simple as HTML. This gap has now been filled by XML.

The development of XML started in 1996 at Sun Microsystems. Jon Bosak with his team began work on a project for remoulding SGML. They took the best of SGML and produced something to be powerful, but much simpler to use.

The World Wide Web Consortium also contributes to the creation and development of the standard for XML. The specifications for XML were laid down in just 26 pages, compared to the 500+ page specification that define SGML.

Difference between HTML and XML

1. XML is designed to carry data.

XML describes and focuses on the data while HTML only displays and focuses on how data looks. HTML is all about displaying information but XML is all about describing information. In current scenario XML is the most common tool for data manipulation and data transmission.

XML is used to store data in files and for sharing data between diverse applications. Unlike HTML document where data and display logic are available in the same file, XML hold only data. Different presentation logics could be applied to display the xml data in the required format. XML is the best way to exchange information.

2. XML is Free and Extensible.

3. XML tags are not predefined. User

XML and JAXP

must "invent" his tags.

The tags used to mark up HTML documents and the structure of HTML documents is predefined. The author of HTML documents can only use tags that are defined in the HTML standard (like `<p>`, `<h1>`, etc.).

XML allows the user to define his own tags and document structure.

4. XML Tags are Case Sensitive

Unlike HTML, XML tags are case sensitive. In HTML the following will work:

```
<Message>This is incorrect</message>
```

In XML opening and closing tags must therefore be written with the same case:

```
<message>This is correct</message>
```

5. XML Elements Must be Properly Nested

Improper nesting of tags makes no sense to XML.

In HTML some elements can be improperly nested within each other like this:

```
<b><i>This text is bold and italic</b></i>
```

In XML all elements must be properly nested within each other like this:

```
<b><i>This text is bold and italic</i></b>
```

6. XML is a Complement to HTML, not a replacement for HTML.

It is important to understand that XML is not a replacement for HTML. In Web development it is most likely that XML will be used to describe the data, while HTML will be used to format and display the same data.
Syntax Rules for XML

The syntax rules for XML are very simple and

strict. These are easy to learn and use. Because of this, creating software that can read and manipulate XML is very easy. Xml enables a user to create his own tags.

Note - XML documents use a self-describing and simple syntax

Let's develop a simple XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u
tonight</Body>
</E-mail>
```

The XML declaration:

The XML declaration should always be included in the first line of the xml document. It defines the XML version and the character encoding used in the document. In this case the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Root Element:

The next line defines the first element of the document. It is called as the root element

```
<E-mail>
```

Child Elements:

Next 4 lines describe the four child elements of the root (To, From, Subject and Body).

```
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u
tonight</Body>
```

XML and JAXP

And finally the last line defines the end of the root element.

```
</E-mail>
```

You may feel from this example that the XML document contains an E-mail To Rohan From Amit. Don't you agree that XML is quite self-descriptive?

Now let's discuss its syntax-rules which are very simple to learn.

All XML elements must have a closing tag

In XML all the elements must have a closing tag like this:

```
<To>Rohan</To>  
<From>Amit</From>
```

XML tags are case sensitive

XML tags are case sensitive. The tag <To> is different from the tag <to>. Hence the opening and closing tags must be written with the same case:

```
<To>Rohan</To>  
<to>Rohan</to>
```

XML Elements Must be Properly Nested

Improper nesting of tags makes no sense to XML. In XML all elements must be properly nested within each other like this in a logical order:

```
<b><i>Hi , how are you.....</i></b>
```

XML Documents Must Have a Root Element

All XML documents must contain a single tag pair to define a root element. All other elements must be written within this root element. All elements can have sub elements called as child elements. Sub elements must be correctly nested within their parent element:

```
<root>  
  <child>  
    <subchild>.....</subchild>  
  </child>  
</root>
```

Always Quote the XML Attribute Values

In XML the attribute value must always be quoted. XML elements can have attributes in name/value pairs just like in HTML. Just look the two XML documents below.

The error in the first document is that the date and version attributes are not quoted .

```
<?xml version=1.0 encoding="ISO-8859-1"?>  
<E-mail date=12/11/2002/>
```

The second document is correct:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<E-mail date="12/11/2002"/>
```

With XML, White Space is preserved

With XML, the white space in a document is preserved.

So a sentence likes this: **Hello** **How are you**, will be displayed like this:

```
Hello                      How are you,
```

Comments in XML

The syntax for writing comments in XML is similar to that of HTML.

```
<!-- This is a comment -->
```

XML Elements

XML Elements are extensible. They have

XML and JAXP

relationships. They have simple naming rules.

XML Elements are Extensible
XML documents can be extended to carry more information.

Look at the following XML example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Body>Be ready for a cruise...i will catch u tonight</Body>
</E-mail>
```

Let's suppose that we create an application to fetch data from the above XML document and produce this output:

E-mail

To: Rohan
From: Amit

Be ready for a cruise...i will catch u tonight

Now, the author wants to add a new feature (let it be a subject line). He can easily achieve it by adding one more tag ie..<Subject>in the xml document. So the new modified xml document will look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u tonight</Body>
</E-mail>
```

Now the new generated output will look like this:

E-mail

To: Rohan
From: Amit

Subject: Surprise....

Be ready for a cruise...i will catch u tonight

XML Elements have Relationships

Elements in a xml document are related as parents and children.

Imagine that this xml document is a description of e-mail:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u tonight</Body>
</E-mail>
```

Here, E-mail is the root element while To, From, Subject and Body are the child elements of the E-mail. Here, E-mail is the parent element of To, From, Subject and Body. To, From, Subject and Body are siblings (or sister elements) because they have the same parentage. Hence, all the XML Elements have Relationships.

XML Element Naming conventions:

XML elements must follow these naming conventions:

Names must not start with a number or punctuation character but it can contain letters, numbers, and other characters without spaces.

Names must not start with the letters xml (or XML, or Xml, etc).

XML Attributes

XML elements can have attributes in the start tag, just like HTML. Attributes are used to provide additional information about elements. **Attributes often provide information that is not a part of the data.** In the example below, the file type is irrelevant to the data, but

XML and JAXP

important to the software that wants to manipulate the element:

```
<file type="gif">roseindia.gif</file>
```

Use the quote styles: "red" or 'red'

Attribute values must always be enclosed in quotes. Use either single or double quotes eg..

```
<color="red">
```

or like this:

```
<color='red'>
```

Note: If the attribute value itself contains double quotes it is necessary to use single quotes, like in this example:

```
<name='Rose "India" Net'>
```

Note: If the attribute value itself contains single quotes it is necessary to use double quotes, like in this example:

```
<name="Rose 'India' Net">
```

Use of Elements vs. Attributes

If you start using attributes as containers for XML data, you might end up with the documents that are both difficult to maintain and manipulate. So the user should use elements to describe the data. Use attributes only to provide data that is not relevant to the reader. Only metadata (data about data) should be stored as attributes, and that data itself should be stored as elements.

This is not the way to use attributes eg..

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail To="Rohan" From="Amit"
Subject="Surprise...."
<Body>Be ready for a cruise...i will catch u
tonight</Body>
</E-mail>
```

Try to avoid using attributes in few of the situations.

Lot of problems occur with using attributes values. They are not easily expandable and cannot contain multiple values. They are not easy to test against a Document Type Definition and are also unable to describe their structure. Becomes more irritating, because of its difficulty to get manipulated by program code.

Here is an example, demonstrating how elements can be used instead of attributes. The following three XML documents contain exactly the same information. A date attribute is used in the first, a date element is used in the second, and an expanded date element is used in the third:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail date="15/05/07">
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u
tonight</Body>
</E-mail>
```

First xml document contains date as a attribute which can not be further extended. But date used a element in second document makes it more flexible.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail >
<date="15/05/07">
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u
tonight</Body>
</E-mail>
```

Second xml document can be further extended as.

XML and JAXP

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail >
<date>
  <day>12</day>
  <month>11</month>
  <year>99</year>
</date>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u
tonight</Body>
</E-mail>
```

XML Validation

XML with correct syntax is Well Formed XML. XML validated against a DTD or a Schema is a Valid XML.

Well Formed XML Documents

A "Well Formed" XML document has correct XML syntax. A "Well Formed" XML document is a document that conforms to the XML syntax rules which were described previously.

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must always be quoted

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u
tonight</Body>
</E-mail>
```

Valid XML Documents:

A "Valid" XML document is a "Well Formed" XML document, which also conforms to the rules of a Document Type Definition (DTD) or a XML

Schema.

The following xml document is validated against a DTD , notice the highlighted text.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE e-mail SYSTEM "Internale-
mail.dtd">
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u
tonight</Body>
</E-mail>
```

XML DTD

A DTD defines the legal elements of an XML document. The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.

XML Schema

XML Schema is an XML based alternative to DTD .W3C supports an alternative to DTD called XML Schema.

Designing XML DTD

1. Introduction to DTD
2. DTD - XML Constituent Components
3. DTD Elements
4. DTD Attributes
5. DTD Entities

1. Introduction to DTD:

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.

A DTD can be defined inside a XML document, or a external reference can be declared.

Internal DTD

If the DTD is defined inside the XML

XML and JAXP

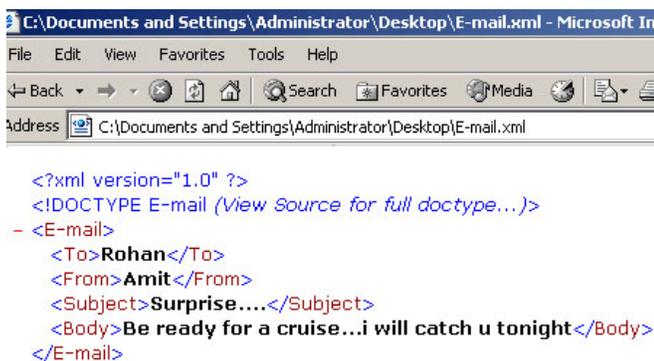
document, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

Example of a XML document with an internal DTD: E-mail.xml

```
<?xml version="1.0"?>
<!DOCTYPE E-mail[
  <!ELEMENT E-mail (To,From,subject,Body)>
  <!ELEMENT To (#PCDATA)>
  <!ELEMENT From (#PCDATA)>
  <!ELEMENT Subject (#PCDATA)>
  <!ELEMENT Body (#PCDATA)>
]>
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u tonight</Body>
</E-mail>
```

Open the file E-mail.xml in a web-browser.



```
<?xml version="1.0" ?>
<!DOCTYPE E-mail (View Source for full doctype...)>
- <E-mail>
  <To>Rohan</To>
  <From>Amit</From>
  <Subject>Surprise....</Subject>
  <Body>Be ready for a cruise...i will catch u tonight</Body>
</E-mail>
```

External DTD

If the DTD is defined in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element SYSTEM
"filename">
```

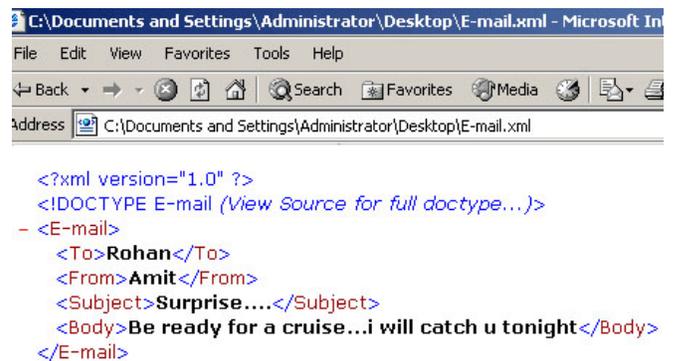
This is the same XML document as above,(but with an external DTD) : E-mail.xml

```
<?xml version="1.0"?>
<!DOCTYPE E-mail SYSTEM "E-mail.dtd">
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...i will catch u tonight</Body>
</E-mail>
```

And this is the file "E-mail.dtd" which contains the following DTD:

```
<!ELEMENT E-mail (To,From,subject,Body)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT From (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Body (#PCDATA)>
```

Open the file E-mail.xml in a web-browser.



```
<?xml version="1.0" ?>
<!DOCTYPE E-mail (View Source for full doctype...)>
- <E-mail>
  <To>Rohan</To>
  <From>Amit</From>
  <Subject>Surprise....</Subject>
  <Body>Be ready for a cruise...i will catch u tonight</Body>
</E-mail>
```

Importance of a DTD:

1. With a DTD, a XML file carries a description of its own format.
2. With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
3. User application can use a standard DTD to verify that the data he receives from the outside world is valid.
4. User can also use a DTD to verify his own data.

XML and JAXP

2. DTD - XML Constituent

DTDs are made up by the following integrants:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

Elements

Elements are the main constituent components of both XML documents.

Elements can contain text, other elements, or be empty.

```
<To>Rohan</To>  
<From>Amit</From>
```

Attributes

Attributes provide extra information about elements. Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

```

```

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a "/".

Entities:

Entities are expanded when a document is parsed by a XML parser. Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag, the greater than sign (>) that defines the end of a XML tag.

The following entities are predefined in XML:

Entity	References	Character
<	<	
>	>	
&		&
"		"
'		'

PCDATA:

PCDATA means **parsed character data**. It can be thought as the character data (text) found between the start tag and the end tag of a XML element. PCDATA is a text to be parsed by a parser. The text is checked by the parser for entities and markup.

Tags inside the text will be treated as markup and entities will be expanded. However, parsed character data should not contain any &, <, or > characters. These should be represented by the &, <, and > entities, respectively.

CDATA:

CDATA is character data that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

3. DTD-Elements

In a DTD, elements are declared with an ELEMENT declaration.

Declaring Elements: syntax

In a DTD, XML elements are declared with the following syntax:

```
<!ELEMENT element-name category>  
or  
<!ELEMENT element-name (element-  
content)>
```

Empty Elements

Empty elements are declared with the keyword EMPTY inside the parentheses.

```
<!ELEMENT element-name EMPTY>
```

XML and JAXP

DTD Example: `<!ELEMENT br EMPTY>`

In XML document:

```
<br />
```

Elements with Parsed Character Data
Elements with only parsed character data are declared with `#PCDATA` inside the parentheses:

```
<!ELEMENT element-name (#PCDATA)>
```

DTD Example:

```
<!ELEMENT To (#PCDATA)>  
<!ELEMENT From (#PCDATA)>
```

Elements with Data
Elements declared with the keyword `ANY`, can contain any combination of parsable data:

```
<!ELEMENT element-name ANY>
```

DTD Example:

```
<!ELEMENT E-mail (To,From,Subject,Body)>  
<!ELEMENT To (#PCDATA)>  
<!ELEMENT From (#PCDATA)>
```

Elements with Children (sequences)
Elements with one or more children are declared with the name of the children elements inside the parentheses as:

```
<!ELEMENT element-name (child1)>  
or  
<!ELEMENT element-name (child1,child2,...)>
```

DTD Example:

```
<!ELEMENT E-mail (To,From,Subject,Body)>
```

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared. Children can have

children. The full declaration of the "E-mail" element is:

```
<!ELEMENT E-mail (To,From,Subject,Body)>  
<!ELEMENT To (#PCDATA)>  
<!ELEMENT From (#PCDATA)>  
<!ELEMENT Subject (#PCDATA)>  
<!ELEMENT Body (#PCDATA)>
```

Declaring Only One Occurrence of an Element

```
<!ELEMENT element-name (child-name)>
```

DTD Example:

```
<!ELEMENT color (Fill-Red)>
```

The example above declares that the child element "Fill-Red" must occur once, and only once inside the "color" element.

Declaring Minimum One Occurrence of an Element

```
<!ELEMENT element-name (child-name+)>
```

DTD Example:

```
<!ELEMENT color (Fill-Red+)>
```

The '+' sign in the example above declares that the child element "Fill-Red" must occur one or more times inside the "color" element.

Declaring Zero or More Occurrences of an Element

```
<!ELEMENT element-name (child-name*)>
```

DTD Example:

```
<!ELEMENT color (Fill-Red*)>
```

The '*' sign in the example above declares that the child element "Fill-Red" can occur zero or more times inside the "color" element.

XML and JAXP

Declaring Zero or One Occurrence of an Element

```
<!ELEMENT element-name (child-name?)>
```

DTD Example:

```
<!ELEMENT color (Fill-Red?)>
```

The '?' sign in the example above declares that the child element "Fill-Red" can occur zero or one time inside the "color" element.

Declaring either/or Content

DTD Example:

```
<!ELEMENT E-mail  
(To,From,Subject,(Message|Body))>
```

The example above declares that the "E-mail" element must contain a "To" element, a "From" element, a "Subject" element, and either a "Message" or a "Body" element.

Declaring Mixed Content

DTD Example:

```
<!ELEMENT E-mail(#PCDATA|To|From|Subject|Body)*>
```

The example above declares that the "E-mail" element can contain zero or more occurrences of a parsed character data, "To", "From", "Subject", or "Body" elements.

4. DTD-Attributes

In a DTD, attributes are declared with an ATTLIST declaration.

Declaring Attributes

The ATTLIST declaration defines the element having a attribute with attribute name, attribute type, and attribute default value. An attribute declaration has the following syntax:

```
<!ATTLIST element-name attribute-name  
attribute-type default-value>
```

DTD example:

```
<!ATTLIST receipt type CDATA "check">
```

XML example:

```
<receipt type="check" />
```

Attribute-type

The attribute-type can be one of the following:

Type Description

CDATA The value is character data (en1|en2|..) The value must be one from an enumerated list

ID The value is a unique id

IDREF The value is the id of another element

IDREFS The value is a list of other ids

NMTOKEN The value is a valid XML name

NMTOKENS The value is a list of valid XML names

ENTITY The value is an entity

ENTITIES The value is a list of entities

NOTATION The value is a name of a notation

xml The value is a predefined xml value

Default-value

The default-value can be one of the following:

Value Explanation

value The default value of the attribute

#REQUIRED The attribute is required

#IMPLIED The attribute is not required

#FIXED value The attribute value is fixed

A Default Attribute Value

DTD Example:

```
<!ELEMENT Scale EMPTY>
```

```
<!ATTLIST Scale length CDATA "0">
```

In the example above, the DTD defines a

XML and JAXP

"Scale" element to be empty with a "length" attribute of type CDATA . If no length is specified, it has a default value of 0.

Valid XML:

```
<Scale length = "100" />
```

REQUIRED Syntax

```
<!ATTLIST element-name attribute_name  
attribute-type #REQUIRED>
```

DTD Example

```
<!ATTLIST person number CDATA  
#REQUIRED>
```

Valid XML:

```
<person id = "5677" />
```

Invalid XML:

```
<person />
```

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

IMPLIED Syntax

```
<!ATTLIST element-name attribute-name  
attribute-type #IMPLIED>
```

DTD Example

```
<!ATTLIST emergency no. CDATA #IMPLIED>
```

Valid XML:

```
<emergency no. = "555-667788" />
```

Valid XML:

```
<emergency />
```

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

FIXED Syntax

```
<!ATTLIST element-name attribute-name  
attribute-type #FIXED "value">
```

DTD Example

```
<!ATTLIST Client CDATA #FIXED  
"RoseIndia">
```

Valid XML:

```
<Client = "RoseIndia" />
```

Invalid XML:

```
<Client = "LotusIndia" />
```

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Enumerated Attribute Values

Syntax

```
<!ATTLIST element-name attribute-name  
(en1|en2|..) default-value>
```

XML and JAXP

DTD Example

```
<!ATTLIST receipt type (check|cash) "cash">
```

XML example:

```
<receipt type="check" />  
or  
<receipt type="cash" />
```

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

5. DTD-Entities

Entities are variables used to define shortcuts to standard text or special characters. Entity references are references to entities. Entities can be declared internally or externally.

Internal Entity Declaration
Syntax

```
<!ENTITY entity-name "entity-value">
```

DTD Example:

```
<!ENTITY name "Amit">  
<!ENTITY company "RoseIndia">
```

XML example:

```
<Profile>&name;&company;</Profile>
```

Note: An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

An External Entity Declaration
Syntax

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

DTD Example:

```
<!ENTITY name SYSTEM "http://  
www.roseindia.net/entities.dtd">  
<!ENTITY company SYSTEM "http://  
www.roseindia.net/entities.dtd">
```

XML example:

```
<Profile>&name;&company;</Profile>
```

Hibernate with Annotation

In the Dec 2007 issue of Java Jazz Up magazine, we have discussed a lot of details about the Hibernate. In the current issue we are taking you further in the same direction but this time we are talking about the annotations with the hibernate.

Hibernate needs a metadata to govern the transformation of data from POJO to database tables and vice versa. Most commonly XML file is used to write the metadata information in Hibernate

Annotations : A Brief Overview

The Java 5 version has introduced a powerful way to provide the metadata to the JVM. The mechanism is known as Annotations. Annotation is the java class which is read through reflection mechanism during the runtime by JVM and does the processing accordingly. The Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development. This also reduces the management of different files for the metadata and java code.

Prerequisites for setting up a project :

1. Make sure to have Java 5.0 or a higher version.
2. Hibernate Core 3.2.0GA and above.
3. Download and add the Hibernate-Annotations jar file in the project workspace.

Let's start with developing an Application :

Let us assume we have a table Employee which has only two columns i.e ID and Name. In hibernate core to achieve the mapping for the above employee table the user should create the following files

1. Utility file for configuring and building the session factory.

2. Hibernate.cfg.xml or any other Datasource metadata file
3. Employee POJO object.
4. Real application file which has the actual logic manipulate the POJO

Note:- Annotations are only the step to remove the hbm.xml file so all the steps remain same only some modifications in some part and adding the annotation data for the POJO.

Please note that coming example will use the Employee table from the database. The SQL query to create the employee table is as follows :-

```
Create table Employee( ID int2 PRIMARY KEY, NAME n varchar(30) );
```

Step 1:- Creating A simple Utility Class

The following is the code given for the utility class for sessionFactory

```
package net.roseindia;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
        }

        catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);

            throw new ExceptionInInitializerError(ex);
        }
    }
}
```

Hibernate with Annotation

```
public static SessionFactory
getSessionFactory() {
    return sessionFactory;
}
}
```

If you see the only change in the file for the annotations is that we simply use `AnnotationConfiguration()` class instead of the `Configuration()` class to build the `sessionFactory` for the hibernate.

Step 2: Creating the Hibernate.cfg.xml

The `Hibernate.cfg.xml` is the configuration file for the datasource in the Hibernate world. The following is the example for the configuration file.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<!-- Database connection settings -->
<property
name="connection.driver_class">com.mysql.jdbc.Driver</
property>
<property
name="connection.url">jdbc:mysql://
localhost:3306/test</property>
<property
name="connection.username">root</
property>
<property
name="connection.password">root</
property>
<!-- JDBC connection pool (use the built-in)
-->
<property name="connection.pool_size">1
</property>
<!-- SQL dialect -->
<property name="dialect">
org.hibernate.dialect.MySQLDialect
</property>
<!-- Enable Hibernate's automatic session
context management -->
```

```
<property
name="current_session_context_class">thread
</property>
<!-- Disable the second-level cache -->
<property name="cache.provider_class">
org.hibernate.cache.NoCacheProvider
</property>
<!-- Echo all executed SQL to stdout -->
<property name="show_sql">>true
</property>
<!-- Drop and re-create the database
schema on startup -->
<property name="hbm2ddl.auto">none
</property>

<mapping class="com.roseindia.Employee"/>

</session-factory>
</hibernate-configuration>
```

The only change is that, we are just telling the compiler that instead of any resource get the metadata for the mapping from the POJO class itself like in this case it is `net.roseindia.Employee`.

Note: Using annotations does not mean that you cannot give the `hbm.xml` file mapping. You can mix annotations and `hbm.xml` files mapping for different Pojo but you cannot mix the mappings for the same POJO in both ways.

Step 3: Developing a simple POJO

The major changes occurred only in the POJO files if you want to use the annotations as this is the file which will now contain the mapping of the properties with the database.

The following is the code for the `Employee` POJO.

```
package net.roseindia;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
```

Hibernate with Annotation

```
@Table(name = "employee")
public class Employee implements
Serializable {
    public Employee() {

    }
    @Id
    @Column(name = "id")
    Integer id;

    @Column(name = "name")
    String name;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

In the EJB world the entities represent the persistence object. This can be achieved by @Entity at the class level. @Table(name = "employee") annotation tells the entity is mapped with the table employee in the database. Mapped classes *must* declare the primary key column of the database table. Most classes will also have a Java-Beans-style property holding the unique identifier of an instance. The @Id element defines the mapping from that property to the primary key column. @Column is used to map the entities with the column in the database.

Step 4: Real application file which has the actual logic manipulate the POJO

The following code demonstrates how to store an entity in the database. The code is identical as you have used in the Hibernate applications.

```
package net.roseindia;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

public class Example1 {

    /**
     * @param args
     */
    public static void main(String[] args)
throws Exception {
        /** Getting the Session Factory and
        session */
        SessionFactory session =
        HibernateUtil.getSessionFactory();
        Session sess =
        session.getCurrentSession();
        /** Starting the Transaction */
        Transaction tx = sess.beginTransaction();
        /** Creating POJO */
        Employee pojo = new Employee();
        pojo.setId(new Integer(5));
        pojo.setName("XYZ");
        /** Saving POJO */
        sess.save(pojo);
        /** Committing the changes */
        tx.commit();
        System.out.println("Record Inserted");
        /** Closing Session */
        session.close();
    }
}
```

Output:

```
Record
```

```
Inserted.
```

This article tried to show the basic configurations for the Hibernate annotations. The above example will only add one record in the Database using the hibernate annotations.

Introduction to Ant

Ant is a platform-independent build tool that specially supports for the Java programming language. It is written purely in Java. Ant is a powerful technique that helps the developers to convert their developmental structures in deployment structures.

It allows the developer to automate the repeated process involved in the development of J2EE application. Developers can easily write the script to automate the build process like compilation, archiving and deployment.

A build process is an essential part of any development cycle because it removes the gap between the development, integration, and test environments. It also helps to remove other issues while deploying such as compilation, classpath, or properties that cost many projects time and money. Ant is a free tool under the GNU License and is freely available at <http://jakarta.apache.org/ant/>.

There are some useful commands described in the table. These commands are built in the Ant distribution.

Command	Description
Ant	Used to execute another ant process from within the current one.
Copydir	Used to copy an entire directory.
Copyfile	Used to copy a single file.
Delete	Deletes either a single file or all files in a specified directory and its sub-directories.
Deltree	Deletes a directory with all its files and subdirectories.
Get	Gets a file from an URL.
Jar	Jars a set of files.
Java	Executes a Java class within the running (Ant) VM or forks another VM if specified.

Javac	Compiles a source tree within the running (Ant) VM.
Mkdir	Makes a directory.
Property	Sets a property (by name and value), or set of properties (from file or resource) in the project.
Rmic	Runs the rmic compiler for a certain class.
Exec	Executes a system command. When the os attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.

Ant Data Types

There are some data types used by the ant tasks. These are described in a table given below:

Data Types	Description
Argument	It passes command-line arguments to programs that you invoke from an Ant buildfile.
Environment	It specifies environment variables to pass to an external command or program that you execute from an Ant buildfile.
Filelist	It defines a named list of files that do not necessarily need to actually exist.
Fileset	It defines a named list of files that must actually exist.
Patternset	It groups a set of patterns together.
Filterset	It groups a set of filters together.

Introduction to Ant

Path **It specifies paths (such as a classpath) in a way that is portable between operating systems.**

Let's develop a Build.xml :

In Ant, all the command line tasks used for deploying an application are represented by simple XML elements. It accepts instructions in the form of XML documents thus it is extensible and easy to maintain. The Ant installation comes with a JAXP-Compliant XML parser, that means the installation of an external XML parser is not necessary.

In this section, a simple Ant example is shown below which is followed by a set of instructions that indicates how to use Ant.

Simple build process with Ant (build.xml):

```
<?xml version="1.0"?>
<project name="antCompile"
default="deploy" basedir=".">
<target name="init">
  <property name="sourceDir"
value="src"/ >
  <property name=" classDir "
value="build" />
  <property name="deployJSP" value="/
web/deploy/jsp" />
</target>

<target name="clean" depends="init">
  <deltree dir="{ classDir }" />
</target>
<target name="prepare" depends="clean">
  <mkdir dir="{ classDir }" />
</target>

<target name="compile"
depends="prepare">
  <javac srcdir="{sourceDir}" destdir="{
classDir }" />
</target>

<target name="deploy"
depends="compile,init">
```

```
<copydir src="{jsp}"
dest="{deployJSP}"/>
</target>
</project>
```

Now, let's understand one by one each tag of this XML file.

1. <?xml version="1.0"?>

Since Ant build files are XML files so the document begins with an XML declaration that specifies which version of XML is in use.

2. **<project name="antCompile" default="deploy" basedir=".">**

The root element of an Ant build file is the project element that contains information about the overall project that is to be built. It has three attributes.

- **name:** It defines the name of the project that can be any combination of alphanumeric characters that constitute valid XML.
- **default:** It references the default target that is to be executed, and when no target is specified. Out of these three attributes default is the only required attribute.
- **basedir:** It is treated as the base directory from which the relative references contained in the build file are retrieved. Each project can have only one basedir attribute.

3. **<target name="init">**

or

<target name="clean" depends="init">

The target element is used as a wrapper for sequences of actions. It contains four attributes: name, if, unless, and depends. Ant requires the **name** attribute, but the other three attributes are optional.

name: The name of the target is used to reference it from elsewhere, so that it can be

Introduction to Ant

referenced from elsewhere, either externally from the command line, or internally via the **depends** keyword, or through a direct call.

depends: depends contains the list of the targets that must be executed prior to executing this target. For example in the second line, the **clean** task will not start until the **init** task has completed.

if: This is a useful attribute which allows one to add a conditional attribute to a target based on the value of a property. The **if** will execute when the property value is set.

unless: This is the converse of **if**. The targets' contents will be executed if the value is not set (to any value).

The **init** target from the simple example contains three lines of property commands as shown here:

```
<property name="sourceDir" value="src" />
<property name="classDir" value="build" />
<property name="deployJSP" value="/
web/deploy/jsp" />
```

The **property** element allows the declaration of commonly used directories or files that are like user-definable variables available for use within an Ant build file. The name attribute specifies the name of the directory or file as a logically and the value attribute specifies the desired value of the property.

For example, in the markup shown above "sourceDir" has been assigned the value "src".

4. Then, you can have the reference of these specified variables later in the Ant file to obtain the value for these tags using `${dir name}`.

For example two other commands present in the above buildfile are:

```
<deltree dir="${classDir}" />
<mkdir dir="${classDir}" />
```

The first command removes the entire tree contained under the **classDir**. The second command makes a directory again using the

mkdir element.

5. <target name="compile" depends="prepare">

```
<javac srcdir="${sourceDir}" destdir="${
classDir }"/>
</target>
```

This section of a build file covers the compilation process. The **javac** element, as used above is a command. It requires a source directory (the input location of the .java files) and a destination directory (the output location of the .classes file). It is important to note that all directories must either exist to run the ant command or be created using the **mkdir** command. The example above will compile all the **.java** files in the directory specified by the **src** property and placing the resultant .class files in the directory specified by the **build** property.

6. <target name="deploy" depends="compile,init">

```
<copydir src="${jsp}" dest="${deployJSP}"/
>
</target>
```

Once the compile task has been complete, at last the **deploy** task will perform the copy operation to copy all JSP files from the source directory to a deployment directory using the **copydir** command.

Struts2 Data Tags

Struts2 Data Tags

Apache Struts is an open-source framework used to develop Java web applications. We started introducing struts generic tags in the November issue. In this section, we will continue further with the data tags (generic tags) provided with struts 2 framework and the rest will be included in the subsequent issues of the magazine.

Just download the zip file

"struts2datatags.zip" from any link given below of each page of this article, unzip it and copy this application to the webapps directory of Tomcat. Start tomcat and write `http://localhost:8080/struts2datatags/index.jsp` to the address bar. You can examine the result of each tag from this page.



Generic Tags

2. Data Tags

- ◆ [Action Tag \(Data Tag\) Example](#)
- ◆ [Bean Tag \(Data Tag\) Example](#)
- ◆ [Date Tag \(Data Tag\) Example](#)
- ◆ [Include Tag \(Data Tag\) Example](#)
- ◆ [Param Tag \(Data Tag\) Example](#)
- ◆ [Set Tag \(Data Tag\) Example](#)
- ◆ [Text Tag \(Data Tag\) Example](#)
- ◆ [Property Tag \(Data Tag\) Example](#)

1. Action Tag (Data Tag) Example

The action tag is a generic tag that is used to call actions directly from a JSP page by specifying the action name and an optional namespace. The body content of the tag is used to render the results from the Action. Any result processor defined for this action in struts.xml will be ignored, unless the executeResult parameter is specified.

Add the following code snippet into the struts.xml file.

struts.xml

```
<action name="actionTag"
class="net.javajazzup.actionTag">
<result name="success"/>/pages/dataTags/
success.jsp</result>
</action>
```

Create an action class as shown below:

actionTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;

public class actionTag extends ActionSupport
{
    public String execute() throws Exception{
        return SUCCESS;
    }
}
```

Now create a jsp page using `<s:action>` tag as shown in the success.jsp page. The action tag is used to call actions directly from a JSP page by specifying the action name and an optional namespace.

Struts2 Data Tags

success.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Action Tag Example</title>
  </head>
  <body>
    <h2>Action Tag Example</h2>
    <s:action name="success">
<b><i>
The action tag will execute the result and
include it in this page.</i></b></div>
    </s:action>
  </body>
</html>
```

Output of the success.jsp



Action Tag Example

The action tag will execute the result and include it in this page.

2. Bean Tag (Data Tag) Example

The Bean tag is a generic tag that is used to instantiate a class that conforms to the JavaBeans specification. This tag has a body, which can contain a number of Param elements to set any mutator methods on that class. If the id attribute is set on the BeanTag, it will place the instantiated bean into the stack's Context.

Add the following code snippet into the struts.xml file.

struts.xml

```
<action name="beanTag"
class="net.javajazzup.beanTag">
  <result name="success">/pages/dataTags/
beanTag.jsp</result>
</action>
```

Create an action class as shown below:

```
beanTag.java
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;

public class beanTag extends ActionSupport {
  public String execute() throws Exception{
    return SUCCESS;
  }
}
```

Create a simple java bean as shown:

companyName.java

```
package net.javajazzup;

public class companyName {
  private String name;

  public void setName(String name){
    this.name =name ;
  }

  public String getName(){
    return name;
  }
}
```

Now create a jsp page using <s:bean> and <s:param> tags as shown in the beanTag.jsp page. The bean tag instantiates the "net.roseindia.companyName" class, it conforms to the JavaBeans specification. The id attribute is set on the BeanTag, it places the instantiated bean into the stack's Context. The body of <s:bean> tag contains a param element <s:param name="name">RoseIndia</s:param> which is used to set the value for the setName() method of the "companyName" class and <s:property value="%{name}" /> retrieves that value by calling the getName() method.

Struts2 Data Tags

beanTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Bean Tag Example</title>
  </head>
</body>
```

Output of the beanTag.jsp



3. Date Tag (Data Tag) Example

The date tag allows formatting a Date in a quick and easy way. User can specify a custom format (eg. "dd/MM/yyyy hh:mm"), can generate easy readable notations (like "in hours, 14 minutes"), or can just fall back on a predefined format with key 'struts.date.format' in the properties file. If that key is not defined, it will finally fall back to the default DateFormat.MEDIUM formatting.

Note: If the requested Date object isn't found on the stack, a blank will be returned.

Configurable attributes are:

1. name
2. nice
3. format

Add the following code snippet into the "struts.xml" file.

struts.xml

```
<action name="dateTag"
class="net.javajazzup.dateTag">
<result>/pages/dataTags/dateTag.jsp
</result>
</action>
```

Create an action class as shown below:

dateTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class dateTag extends ActionSupport {
  private Date currentDate;
  public String execute() throws Exception{
    setCurrentDate(new Date());
    return SUCCESS;
  }
  public void setCurrentDate(Date date){
    this.currentDate = date;
  }
  public Date getCurrentDate(){
    return currentDate;
  }
}
```

Now create a jsp page using <s:date> tag as shown in the success.jsp page.

The <s:date name="currentDate" format="dd/MM/yyyy" /> date tag formats a Date in a quick and easy way. Here the "format" parameter specifies a custom format (eg. "dd/MM/yyyy hh:mm") to follow.

The nice parameter is of Boolean type, which decides whether to print out the date nicely or not. By Default it is kept false which prints out date nicely i.e. <s:date name="currentDate" nice="false" /> tag formats a date and similarly <s:date name="currentDate" nice="true" /> does not format a date, it is illustrated in our current jsp page.

Struts2 Data Tags

dateTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Date Tag Example</title>
  </head>
  <body>
    <h2>Current Date Format</h2>
    <table border="1" width="35%">
      <tr>
        <td><b>Date Format</b></td>
        <td><b>Date</b></td>
      </tr>
      <tr>
        <td>Day/Month/Year</td>
        <td><s:date name="currentDate"
format="dd/MM/yyyy" /></td>
      </tr>
      <tr>
        <td>Month/Day/Year</td>
        <td><s:date name="currentDate"
format="MM/dd/yyyy" /></td>
      </tr>
      <tr>
        <td>Month/Day/Year</td>
        <td><s:date name="currentDate"
format="MM/dd/yy" /></td>
      </tr>
      <tr>
        <td>Month/Day/Year Hour<B>:</
B>Minute</td>
        <td><s:date name="currentDate"
format="MM/dd/yy hh:mm" /></td>
      </tr>
      <tr>
        <td>Month/Day/Year Hour<B>:</
B>Minute<B>:</B>Second</td>
        <td><s:date name="currentDate"
format="MM/dd/yy hh:mm:ss" /></td>
      </tr>
      <tr>
        <td>Nice Date (Current Date &
Time)</td>
        <td><s:date name="currentDate"
nice="false" /></td>
      </tr>
      <tr>
        <td>Nice Date</td>
        <td><s:date name="currentDate"
nice="true" /></td>
      </tr>
    </table>
  </body>
</html>
```

```
</tr>
</table>

</body>
</html>
```

Output of the dateTag.jsp :



Current Date Format

Date Format	Date
Day/Month/Year	04/12/2007
Month/Day/Year	12/04/2007
Month/Day/Year	12/04/07
Month/Day/Year Hour:Minute	12/04/07 05:43
Month/Day/Year Hour:Minute:Second	12/04/07 05:43:53
Nice Date (Current Date & Time)	Dec 4, 2007 5:43:53 PM
Nice Date	an instant ago

4. Include Tag (Data Tag) Example

The include tag is a generic tag that is used to include a servlet's output (result of servlet or a JSP page) to the current page.

Add the following code snippet into the struts.xml file.

struts.xml

```
<action name="includeTag"
class="net.javajazzup.includeTag">
<result>/pages/dataTags/includeTag.jsp
</result>
</action>
```

Struts2 Data Tags

Create an action class as shown below:
includeTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class includeTag extends ActionSupport
{
private Date myBirthday;
public String execute() throws Exception{
setMyBirthday(new Date("Jan 12, 1984
11:21:30 AM"));

return SUCCESS;
}
public void setMyBirthday(Date date){
this.myBirthday = date;
}
public Date getMyBirthday(){
return myBirthday;
}
}
```

Create a simple jsp (myBirthday.jsp) that we want to include in our main jsp page ie. includeTag.jsp.

myBirthday.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
<head>
<title>Include Tag Example</title>
</head>
<body>
<b>My Birth Day (Date Format)</b>
<table border="1" width="35%">
<tr>
<td><b>Date Format</b></td>
<td><b>Date</b></td>
</tr>
<tr>
<td>Day/Month/Year</td>
<td><s:date name="myBirthday"
format="dd/MM/yyyy" /></td>
</tr>
<tr>
<td>Month/Day/Year</td>
```

```
<td><s:date name="myBirthday"
format="MM/dd/yyyy" /></td>
</tr>
<tr>
<td>Month/Day/Year</td>
<td><s:date name="myBirthday"
format="MM/dd/yy" /></td>
</tr>
<tr>
<td>Month/Day/Year Hour<B>:</
B>Minute</td>
<td><s:date name="myBirthday"
format="MM/dd/yy hh:mm" /></td>
</tr>
<tr>
<td>Month/Day/Year Hour<B>:</
B>Minute<B>:</B>Second</td>
<td><s:date name="myBirthday"
format="MM/dd/yy hh:mm:ss" /></td>
</tr>
<tr>
<td>Nice Date (Current Date &
Time)</td>
<td><s:date name="myBirthday"
nice="false" /></td>
</tr>
</table>

</body>
</html>
```

Now create a jsp page using <s:include> tag as shown in the includeTag.jsp page. The <s:include value="myBirthday.jsp" /> tag includes another jsp using the value parameter.

includeTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
<head>
<title>Include Tag Example</title>
</head>
<body>
<h2>Include Tag Example</h2>
<s:include value="myBirthday.jsp" />
</body>
</html>
```

Struts2 Data Tags

Output of the includeTag.jsp:



Include Tag Example

My Birth Day (Date Format)

Date Format	Date
Day/Month/Year	12/01/1984
Month/Day/Year	01/12/1984
Month/Day/Year	01/12/84
Month/Day/Year Hour:Minute	01/12/84 11:21
Month/Day/Year Hour:Minute:Second	01/12/84 11:21:30
Nice Date (Current Date & Time)	Jan 12, 1984 11:21:30 AM

5. Param Tag (Data Tag) Example

The param tag is a generic tag that is used to parameterize other tags. For example the include tag and bean tag. The parameters can be added with or without a name as a key.

The param tag has the following two parameters.

1. name (String) - the name of the parameter
2. value (Object) - the value of the parameter

Note: When you declare the param tag, the value can be defined in either a value attribute or as text between the start and end tag. Struts behave a bit different according to these two situations.

Case 1. `<param name="empname">Amit</param>` Here the value would be evaluated to the stack as a `java.lang.String` object.

Case 2. `<param name="empname" value="Vinod"/>` Here the value would be evaluated to the stack as a `java.lang.Object` object.

Add the following code snippet into the `struts.xml` file.

struts.xml

```
<action name="paramTag">
<result>/pages/dataTags/paramTag.jsp
</result>
</action>
```

Now create a jsp page to see the working of the param tags.

paramTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Param Tag Example</title>
  </head>
  <body>
    <h2>Param Tag Example</h2>
    <ui:component>
      <ui:param name="empname">Emp1</
ui:param><br>
      <ui:param name="empname">Emp2</
ui:param><br>
      <ui:param name="empname">Emp3</
ui:param>
    </ui:component>
  </body>
</html>
```

Struts2 Data Tags

Output of paramTag.jsp:



Param Tag Example

Emp1
Emp2
Emp3

6. Set Tag (Data Tag) Example

The set tag is a generic tag that is used to assign a value to a variable in a specified scope. It is useful when you wish to assign a variable to a complex expression and then simply reference that variable each time rather than the complex expression.

Add the following code snippet into the struts.xml file.

struts.xml

```
<action name="setTag">  
<result>/pages/dataTags/setTag.jsp</result>  
</action>
```

Now create a jsp page using `<s:set>` tag as shown in the setTag.jsp page. The set tag is used to assign a value to a variable in a specified scope. The parameters name and value in the tag `<s:set name="technologyName" value="%{\Java}'"/>` acts as the name-value pair. Here we set the parameters as name="technologyName" value="Java".

setTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>  
  
<html>  
  <head>  
    <title>Set Tag Example</title>  
  </head>  
  <body>  
    <h2>Set Tag Example</h2>  
    <s:set name="technologyName"  
value="%{\Java}'"/>  
    Technology Name: <s:property  
value="#technologyName"/>  
  </body>  
</html>
```

Output of the setTag.jsp:



7. Text Tag (Data Tag) Example

The text tag is a generic tag that is used to render a I18n text message. Follow one of the three steps:

1. Keep the message to be displayed in a resource bundle with the same name as the action that it is associated with ie. Create a properties file in the same package as your Java class with the same name as your class, but with. properties extension.
2. If the property file does-not work or the message is not found in the resource

Struts2 Data Tags

bundle, then the body of the tag will be used as default message.

3. If there is no body, then the name of the message will be used.

Add the following code snippet into the struts.xml file.

struts.xml

```
<action name="textTag"
class="net.javajazzup.textTag">
  <result>/pages/dataTags/textTag.jsp</
result>
</action>
```

Create an action class as shown below:
textTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;

public class textTag extends ActionSupport {

  public String execute() throws Exception{
    return SUCCESS;
  }
}
```

Create a property file in the same package where your Java program file (textTag.java) is saved with the name as

package.properties

package.properties

```
webname1 = http://www.javajazzup.com
webname2 = http://www.roseindia.net
webname3 = http://www.newstrackindia.com
```

Now create a jsp page to see the working of the text tags.

The first three tags `<s:text name="webname1">`, `<s:text name="webname2">` and `<s:text name="webname3">` uses the package.properties file to display the text message.

The next tag

`<s:text name="empname">Vinod, Amit, Sushil,</s:text>` uses the body of the tag as a default message.

The last tag `<s:text name="empname"></s:text>` does not have access to the package.properties files nor does it have a body so it uses name of the message to display.

textTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Text Tag Example</title>
  </head>
  <body>
    <h2>Text Tag Example</h2>

    <s:text name="webname1"></s:text><br>
    <s:text name="webname2"></s:text><br>
    <s:text name="webname3"></s:text><br>
    <s:text name="empname">Emp1,Emp2....
    </s:text><br>
    <s:text name="empname"> </s:text>
  </body>
</html>
```

Output of the textTag.jsp:



Text Tag Example

```
http://www.javajazzup.com
http://www.roseindia.net
http://www.newstrackindia.com
Emp1,Emp2....
empname
```

Struts2 Data Tags

8. Property Tag (Data Tag) Example

The property tag is a generic tag that is used to get the property of a value, which will default to the top of the stack if none is specified.

Add the following code snippet into the struts.xml file.

struts.xml

```
<action name="propertyTag"
class="net.javajazzup.propertyTag">
<result>/pages/dataTags/propertyTag.jsp
</result>
</action>
```

Create an action class as shown:

propertyTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;

public class propertyTag extends
ActionSupport {
    public String execute() throws Exception{
        return SUCCESS;
    }
}
```

Create a bean class "companyName" as shown:

companyName.java

```
package net.javajazzup;

public class companyName {
    private String name;

    public void setName(String name){
        this.name =name ;
    }

    public String getName(){
        return name;
    }
}
```

Create a jsp using the tags.

<s:property value="%{name}" /> it prints the result of myBean's getMyBeanProperty() method.

<s:property value="name" default="Default Value" /> it prints the result of companyName's getName() method and if it is null, print 'a default value' instead.

propertyTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
<head>
<title>Property Tag Example</title>
</head>
<body>
<h2>Property Tag Example</h2>
<!-- Example to pick the value through bean
class -->
<s:bean
name="net.javajazzup.companyName"
id="uid">
<s:param name="name">JavaJazzUp</
s:param>
<s:property value="%{name}" /> <br>
</s:bean>
<!-- Default value -->
<s:property value="name"
default="Default Value" />
</body>
</html>
```

Output of the propertyTag.jsp:



Property Tag Example

JavaJazzUp
Default Value

Integrating JSF, Spring and Hibernate

Integrating JSF, Spring and Hibernate

This article explains integrating JSF (MyFaces), Spring and Hibernate to build real world User Login and Registration Application using MySQL as database. This application lets the new user create an account and existing user access the application by user name and password. These three frameworks can be used for different purposes according to the usefulness and power of individual frameworks. Like,

- 1 JSF can be used to implement presentation layer because it fits into the MVC design pattern.
- 2 Spring Framework can be used in the business logic tier to manage business objects, resource management.
- 3 Hibernate can be powerful inside the integration tier. Spring integrates Hibernate very well.

Let's see how these have been utilized to develop our User Login and Registration Application. Development of this application has been divided in various steps to understand it clearly.

1. About the Application
2. Application Architecture
3. Downloading MyFaces and creating web application
4. Adding Spring and Hibernate Capabilities
5. Setting up MySql database and creating tables

Developing Presentation Layer

6. Developing Login and Registration form and backing beans

Developing Business Layer

7. Writing Business Objects

Integration Tier of the Application

8. Implementing Integration tier with Hibernate

Wiring up everything

9. Integrating JSF, Spring and Hibernate

10. Integrating presentation layer

11. Integrating Business Logic/Integration Tier

12. Downloading the full code of this application

1. About the Application

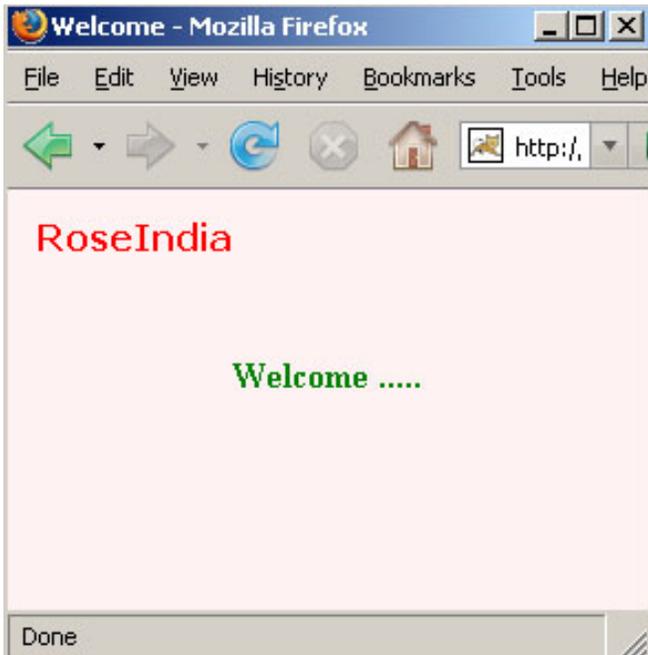
a) User Login Module

In this module, the user is asked to fill its user ID and password to proceed further in the application.



If the user fills correct user name and password then it welcomes the user and displays the page like the following.

Integrating JSF, Spring and Hibernate



If user doesn't fill it correctly then it shows a message indicating wrong entry.



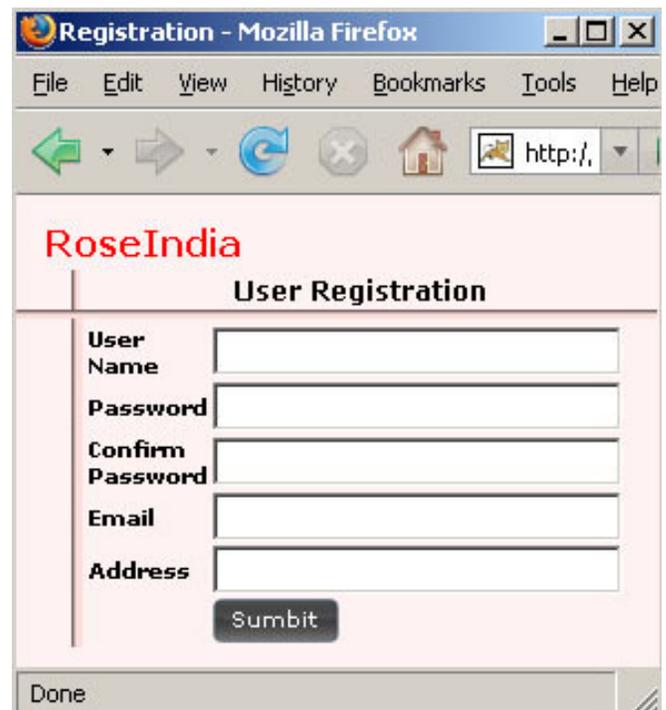
If the user is not registered yet then there is a link "New User?" to register such users.

b) User Registration Module:

In this module, the user is asked to fill the required information like User Name, Password, Email Address and Physical Address. User is asked to reenter the password to confirm it. In this form, all the fields have individual validation checks so that the user should enter correct entries. So for this we have taken care of some points:

1. Make the fields required to fill.
2. Confirm password field.
3. Check for matching the passwords.
4. Email address validation to check the correct format of email address.
5. Individual Messages for every field if there is any mismatch in the passed information.

The registration page can be seen below.



If the user fills any incorrect entry then related message is shown for that particular field, like below:

Integrating JSF, Spring and Hibernate

If the user is already registered with the site then a message flashes as in the figure



If the registration process completes successfully then the screen below appears:



2. Application Architecture

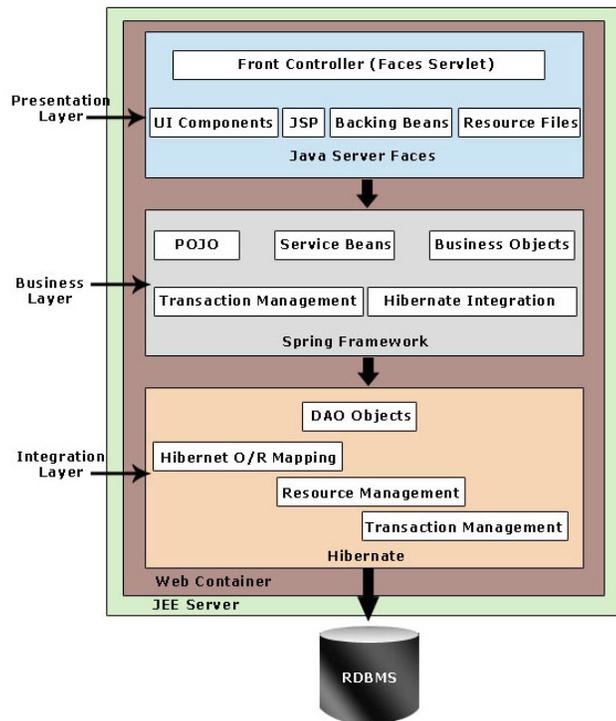
Login and Registration application consists of 3 different layers (tiers):

1. Presentation Layer
2. Business Layer
3. Data Access Layer

In this application the presentation layer, business layer and data access layer are physically located on the same JEE server. Different layers of the application are isolated from each other and connected through well-defined interfaces.

Three tier architecture of the application:

Integrating JSF, Spring and Hibernate



Presentation Layer

JSF is used to build the presentation layer of the application. JSF allows us to create rich GUI for web application. It resolves the technical challenges of creating rich GUI web application. In this layer we have JSP pages consists of JSF components. All the requests to the web server pass through FacesServlet.

Business Layer

The POJO classes and classes to process the business logic are used to create the Business Layer. The POJO classes, with the help of spring framework, create an ideal solution to implement the Business Layer.

Data Access Layer

The data access layer handles all the logic to save and retrieve the data from database. Hibernate O/R mapping tools is an ideal solution for enterprise application of any size. Hibernate handles all the logic to store and retrieve POJO objects. It also handles resource management and transaction management activities.

3. Downloading MyFaces and creating web application

Downloading MyFaces

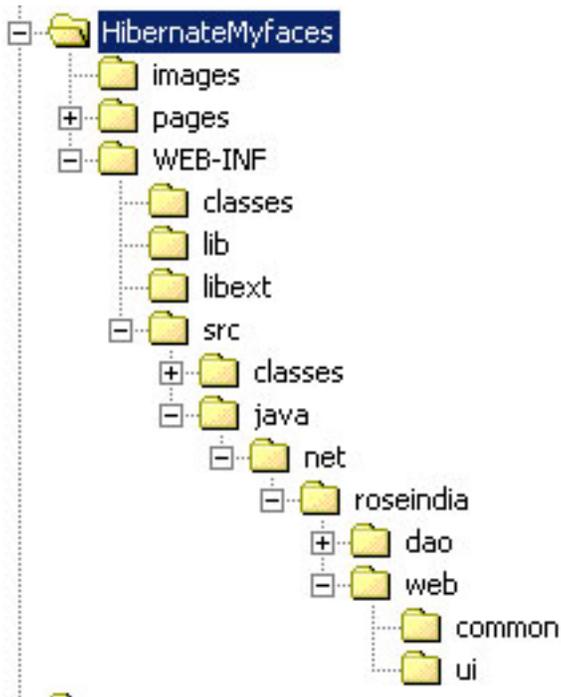
MyFaces can be configured using libraries and configuration files which come with the example applications. The latest version of MyFaces can be downloaded from <http://myfaces.apache.org/download.html>. We have downloaded **tomahawk-examples-1.1.6-bin.zip** from <http://www.apache.org/dyn/closer.cgi/myfaces/binaries/tomahawk-examples-1.1.6-bin.zip> for this example tutorial. The downloaded file will be a zip file named tomahawk-examples-1.1.6-bin.zip. Extract the zip file and you will get 4 war files. Copy myfaces-example-simple-1.1.6.war file in the **webapps** directory of Tomcat which will automatically be expanded in the directory of same name. Now we will use the libraries and configuration files from the exploded application to create our web application.

Creating web application

Web application follows the start directory structure as defined in the JEE (J2EE) specification. Here we are creating the application in the exploded format, you can also create archive (war, ear) and then deploy on the application server.

Following image shows the directory structure of our web application.

Integrating JSF, Spring and Hibernate



Now follow the following steps to create the application:

1. Create a directory with the name of your application (**HibernateMyFaces**) under **webapps** directory of tomcat.
2. Create "**WEB-INF**" directory inside it.
3. Create other directories as shown in the above image.
4. Copy libraries files from "**myfaces-example-simple-1.1.6\WEB-INF\lib**" directory to "**lib**" directory.
5. Create web.xml file under **WEB-INF** directory and add the following content.

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
<!-- Spring context Configuration Begins-->
  <context-param>
    <param-name>log4jConfigLocation</param-name>
    <param-value>/WEB-INF/log4j.properties</param-value>
```

```
</context-param>
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext-hibernate.xml</param-value>
</context-param>
<servlet>
  <servlet-name>context</servlet-name>
  <servlet-class>
    org.springframework.web.context.ContextLoaderServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<!--End Spring configuration -->

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

<!-- Extensions Filter -->
<filter>
  <filter-name>extensionsFilter</filter-name>
  <filter-class>
    org.apache.myfaces.component.html.util.ExtensionsFilter
  </filter-class>

  <init-param>
    <param-name>uploadMaxFileSize</param-name>
    <param-value>100m</param-value>
    <description>Set the size limit for uploaded files.
      Format: 10 - 10 bytes
      10k - 10 KB
      10m - 10 MB
      1g - 1 GB
    </description>
  </init-param>
</init-param>
```

Integrating JSF, Spring and Hibernate

```
<param-name>uploadThresholdSize
</param-name>
  <param-value>100k</param-value>
  <description>Set the threshold size -
files below this limit are stored in memory,
files above this limit are stored on disk.
Format: 10 - 10 bytes
      10k - 10 KB
      10m - 10 MB
      1g - 1 GB
  </description>
</init-param>
</filter>

<filter-mapping>
  <filter-name>extensionsFilter
</filter-name>
  <url-pattern>*.jsf</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>extensionsFilter</filter-
name>
  <url-pattern>/faces/*</url-pattern>
</filter-mapping>
<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>*.jsf</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-
file>
</welcome-file-list>
</web-app>
```

6. Now create **faces-config.xml** and **add the following code.**

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC "-//Sun
Microsystems, Inc.//DTD JavaServer Faces
Config 1.0//EN" "http://java.sun.com/dtd/
web-facesconfig_1_0.dtd" >
```

```
<faces-config>
  <application>
    <locale-config>
      <default-locale>en
    </default-locale>
    </locale-config>
    <message-
bundle>net.roseindia.web.ui.messages
</message-bundle>
  </application>

  <managed-bean>
    <managed-bean-name>Bean
  </managed-bean-name>
    <managed-bean-
class>net.roseindia.web.ui.Bean
</managed-bean-class>
    <managed-bean-scope>session
  </managed-bean-scope>
  </managed-bean>

  <managed-bean>
    <managed-bean-name>CheckValidUser
  </managed-bean-name>
    <managed-bean-class>
      net.roseindia.web.ui.CheckValidUser
    </managed-bean-class>
    <managed-bean-scope>session
  </managed-bean-scope>
  </managed-bean>

  <navigation-rule>
    <from-view-id>/pages/login.jsp
  </from-view-id>
    <navigation-case>
      <from-outcome>reg
    </from-outcome>
      <to-view-id>/pages/
registration.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/pages/
successLogin.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>failure</from-outcome>
      <to-view-id>/pages/login.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <navigation-rule>
```

Integrating JSF, Spring and Hibernate

```
</from-view-id>/pages/registration.jsp</
from-view-id>
<navigation-case>
<from-outcome>success</from-outcome>
<to-view-id>/pages/welcome.jsp
</to-view-id>
</navigation-case>
<navigation-case>
<from-outcome>failure</from-outcome>
<to-view-id>/pages/registration.jsp
</to-view-id>
</navigation-case>
</navigation-rule>
</faces-config>
```

Note: In the next sections we will add the configuration into **web.xml** and **faces-config.xml** files, so the content will change. We advice you to download the full code from our site and then take the files from there.

7. Create **"src"** directory parallel to the **"lib"** directory.

8. Create two directories **"classes"**, **"java"** and **"build.xml"** file within **"src"** folder. Copy the following content into **build.xml** file:

```
<project name="MyFaces, Hibernate and
Spring Integration" basedir=".."
default="all">
<!-- Project settings -->
<property name="project.title"
value="RoseIndia MyFaces, Hibernate and
Spring Integration Tutorial"/>
<property name="project.jar.file"
value="roseindia.jar"/>

<path id="class.path">
<fileset dir="lib">
<include name="**/*.jar"/>
</fileset>
<fileset dir="libext">
<include name="**/*.jar"/>
</fileset>
</path>
<!-- Classpath for Project -->
<path id="compile.classpath">
<pathelement path="lib/myfaces-api-
1.1.5.jar"/>
<pathelement path="lib/myfaces-impl-
```

```
1.1.5.jar"/>
<pathelement path="lib/jstl-1.1.0.jar"/
>
<pathelement path="lib/tomahawk-
1.1.6.jar"/>
<pathelement path="libext/servlet-
api.jar"/>
<pathelement path="classes"/>
<pathelement path="${classpath}"/>
</path>

<!-- Check timestamp on files -->
<target name="prepare">
<tstamp/>
</target>
<!-- Copy any resource or configuration
files -->
<target name="resources">
<copy todir="src/classes"
includeEmptyDirs="no">
<fileset dir="src/java">
<patternset>
<include name="**/*.conf"/>
<include name="**/*.properties"/>
<include name="**/*.xml"/>
</patternset>
</fileset>
</copy>
</target>
<!-- Normal build of application -->
<target name="compile"
depends="prepare,resources">
<javac srcdir="src" destdir="src/
classes" debug="true"
debuglevel="lines,vars,source">
<classpath refid="class.path"/>
</javac>
<jar jarfile="lib/${project.jar.file}"
basedir="src/classes"/>
</target>
<!-- Remove classes directory for clean
build -->
<target name="clean"
description="Prepare for clean build">
<delete dir="classes"/>
<mkdir dir="classes"/>
</target>
<!-- Build Javadoc documentation -->
<target name="javadoc"
```

Integrating JSF, Spring and Hibernate

```
description="Generate Javadoc API docs">
  <delete dir="./doc/api"/>
  <mkdir dir="./doc/api"/>
<javadoc sourcepath="./src/java" destdir="./
doc/api"
  classpath="{servlet.jar}:{jdbc20ext.jar}"
  packagenames="*"
author="true" private="true" version="true"
  windowtitle="{project.title} API
Documentation"
doctitle="&lt;h1&gt;{project.title}
Documentation (Version
{project.version})&lt;/h1&gt;"
  bottom="Copyright &#169;
2002">
  <classpath
refid="compile.classpath"/>
  </javadoc>
</target>
<!-- Build entire project -->
<target name="project"
depends="clean,prepare,compile"/>
  <!-- Build project and create
distribution-->
  <target name="all" depends="project"/>
</project>
```

We will use **ant tool** to build the application, so make sure ant tool is installed on your development machine.

9. Create directory with name **"net"** in **"java"** directory and directory of name **"roseindia"** within **"net"** directory.

10. Create **"classes"** directory within **"WEB-INF"** directory for the class file to be used in the application.

4. Adding Spring and Hibernate Capabilities

In this section, we will add Spring and Hibernate capabilities to our web application. So for this, we will have to add the spring and hibernate

libraries and configuration files to the web application.

Steps to integrate:

Adding Servlets Library

Since we are using ant build tool to compile the application, it is necessary to make **servlets api** library available to ant build tool for compiling the java code. We are adding Servlets library to the project.

1. Create a directory named **libext** under **WEB-INF** directory of your application.

2. Copy **servlet-api.jar** from **common\lib** directory of your **Tomcat** into **libext** directory.

Adding Hibernate Libraries

Now we will download hibernate libraries and add it to the project.

1. Download **hibernate-3.2.4.sp1.zip** or latest version from **http://www.hibernate.org/6.html**. We have downloaded **hibernate-3.2.4.sp1.zip** for this tutorial.

2. **Unzip** the file.

3. **Copy all the jar files** from **hibernate-3.2.4.sp1\hibernate-3.2\lib** into **lib** directory of your application. (The directories **hibernate-3.2.4.sp1\hibernate-3.2** may be different in your case depending on the downloaded version).

4. Copy **hibernate3.jar** from **hibernate-3.2.4.sp1\hibernate-3.2** into **lib** directory of your application.

Adding Spring Libraries

Our application uses spring to manage the beans, so we have to add the spring capabilities to the application.

1. Download **spring-framework-2.0.5-with-dependencies.zip** from **http://www.springframework.org/download**.

2. **Extract the file.**

Integrating JSF, Spring and Hibernate

3. Copy **spring.jar** from spring-framework-2.0.5-with-dependencies\spring-framework-2.0.5\dist directory into **lib** directory of your application.

Now all the libraries are available with us in our application, we will now create database and tables in MySQL to use in our application.

5. Setup MySQL Database

In this section, we will create database and table into MySQL database. Table created here will be used in sample application.

Downloading and Installing MySQL

You can download **MySQL** from **mysql.org**.

Creating Database:

You can create mysql database by issuing the following command:

```
Mysql>create database jsf_hibernate;
```

Creating Table:

To create table in this database you would have to first select the database by use statement.

```
Mysql>use jsf_hibernate
```

Now, you can create table using the following command:

```
mysql>create table users (  
  userId int(11) NOT NULL auto_increment,  
  userName varchar(20) default NULL,  
  userPassword varchar(11) default NULL,  
  userEmail varchar(30) default NULL,  
  userAddress varchar(30) default NULL,  
  PRIMARY KEY (userId)  
);
```

6. User Login and Registration application

Creation of Pages:

This is the first page that appears to the user. In this page user is asked to fill the user name and password to login itself in the site. Both the fields are required to fill. If the user leaves it blank then the message indicating that these

can't be left empty is shown.

Login Page:



login.jsp : The code for above page is given below. For this page **CheckValidUser** backing bean has been used. On the submission of the page **checkUser()** method of the bean is called to check the existence of the user. If the user is registered then user is welcomed to the next page otherwise a message that either user name or password is incorrect.

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>  
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>  
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>  
  
<f:view>  
<html>  
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">  
<head>  
<title>User Login</title>  
<link href="pages/mycss.css" rel="stylesheet" type="text/css"/>  
</head>  
<body ><center>  
  
<h:panelGrid width="100%" columns="1"
```

Integrating JSF, Spring and Hibernate

```
border="0"
  style="padding-left:10px; padding-
top:10px; " styleClass="top_bg">
  <h:dataTable id="dt1" border="0"
cellpadding="0" cellspacing="0" var="ab">
  <h:column>
    <f:facet name="header">
      <h:outputText
value="RoseIndia" styleClass="style4"/>
    </f:facet>
  </h:column>
</h:dataTable>
</h:panelGrid>

<h:panelGrid width="175px" columns="3"
border="0" cellspacing="0" cellpadding="0">
  <h:outputText value=" "/>
  <h:graphicImage id="gi3"
value="images/verticle_line.gif"
width="4" height="18"></
h:graphicImage>
  <h:panelGroup>
    <h:dataTable id="dt2" border="0"
cellpadding="0"
      cellspacing="0"
width="250" var="gh">
      <h:column>
        <f:facet name="header">
          <h:outputText
value="User Login" styleClass="style1"/>
        </f:facet>
      </h:column>
    </h:dataTable>
  </h:panelGroup>

  <h:graphicImage id="gi4"
value="images/horizontal_line.gif"
width="25" height="4" ></h:graphicImage>
  <h:graphicImage id="gi8"
value="images/horizontal_line.gif"
width="5" height="4" ></
h:graphicImage>
  <h:graphicImage id="gi6"
value="images/horizontal_line.gif"
width="260" height="4" ></h:graphicImage>
  <h:outputText value=" "/>
  <h:graphicImage id="gi7"
value="images/verticle_line.gif"
width="5" height="100%" >
</h:graphicImage>
  <h:panelGroup>
    <h:form>
```

```
<h:panelGrid width="75px" columns="2"
border="0">
<f:facet name="header">
  <h:outputText value="User Name or
Password is incorrect"
style="color:red; font-weight: bold;"
rendered="#{CheckValidUser.exist}"/>
</f:facet>
<h:panelGroup>
  <h:outputText value=" "/>
  <h:outputText value="User Name"
styleClass="style2"/>
</h:panelGroup>
<h:panelGroup>
  <h:inputText id="UserName"
value="#{CheckValidUser.userName}"
size="27" required="true"/>
  <f:verbatim><br/></f:verbatim>
  <h:message for="UserName"
styleClass="errors"/>
</h:panelGroup>
<h:panelGroup>
  <h:outputText value="Password"
styleClass="style2"/>
</h:panelGroup> <h:panelGroup>
  <h:inputSecret id="Password"
value="#{CheckValidUser.pwd}"
size="27" required="true"/>
  <f:verbatim><br/></f:verbatim>
  <h:message for="Password"
styleClass="errors"/>
</h:panelGroup>
  <h:outputText value=" "/>
  <h:panelGroup>
    <h:commandButton image="images/
submit_button.gif"
action="#{CheckValidUser.checkUser}"/>
  </h:panelGroup>
</h:panelGrid>
</h:form>
<h:form>
  <h:commandLink value="New User?"
action="reg" styleClass="style3"/>
  </h:form>
</h:panelGroup>
</h:panelGrid>
</center>
</body>
</html>
</f:view>
```

Integrating JSF, Spring and Hibernate

Login Successful Page:

If the user enters correct information then the user is presented with the next page as in the figure below:



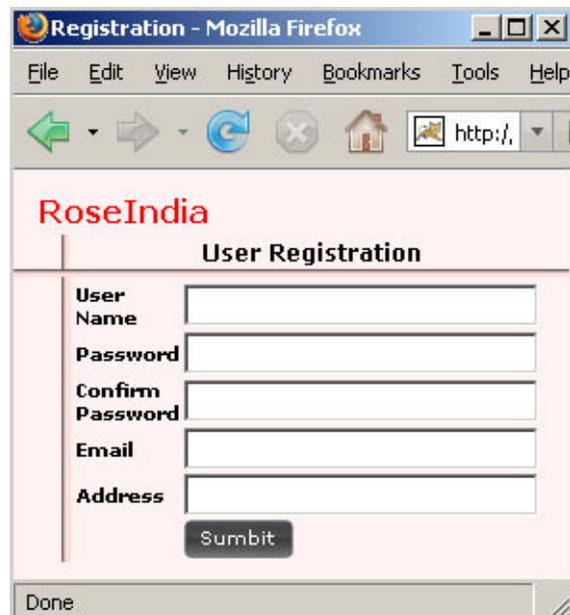
successLogin.jsp: This is the code for above page. This is simple page in which only welcome string is shown to the user.

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<f:view>
<html>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<head>
<title>Welcome</title>
<link href="mycss.css" rel="stylesheet" type="text/css"/>
</head>
<body ><center>
<h:form>
<h:panelGrid width="100%" columns="1" border="0" style="padding-left:10px; padding-top:10px; \ styleClass="top_bg">
<h:dataTable id="dt1" border="0"
```

```
cellpadding="0" cellspacing="0" var="ab">
<h:column>
<f:facet name="header">
<h:outputText value="RoseIndia" styleClass="style4"/>
</f:facet>
</h:column>
</h:dataTable>
</h:panelGrid>
<h:panelGrid width="100%" columns="1" border="0" >
<f:verbatim>&nbsp;</f:verbatim>
<h:outputText value=" "/><h:outputText value=" "/><h:outputText value=" "/>
</h:panelGrid>
<h:outputText value="Welcome ....." style="color:green; font-weight:bold"/>
</h:form>
</center>
</body>
</html>
</f:view>
```

Registration Page:

This page comes to the user if the user is not registered with the site. The user clicks the link named **"New User?"** and so registration page is opened. The figure below is registration page:



Integrating JSF, Spring and Hibernate

registration.jsp : This is the code for above page. The backing bean used for this page is "**Bean**". When the page is submitted then **register()** method is called which checks the user name entered by the user. If the name is already registered then user is informed that the user with name is already registered otherwise a new user object is created and all the fields related with the user is added in that object i.e. user name, password, email, address. Before checking the user existence **validateData()** method is called to check all the fields. If anything doesn't match with the requirement then the same page is again presented to the user with the validation messages. The messages can be customized creating the **MessageFactory** class in which messages from the resource bundle for the specific locale are picked and shown in the page. You can see the code for bean below in the tutorial.

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<f:loadBundle
  basename="net.roseindia.web.ui.messages"
  var="message"/>
<f:view>
<html>
<meta http-equiv="Content-Type"
  content="text/html; charset=iso-8859-1">

<head>
<title>Registration</title>
<link href="mycss.css" rel="stylesheet"
  type="text/css"/>
</head>
<body ><center>

<h:form id="registerForm">
  <h:panelGrid width="100%" columns="1"
  border="0" style="padding-left:10px;
    padding-top:10px; "
  styleClass="top_bg">
    <h:dataTable id="dt1" border="0"
  cellpadding="0" cellspacing="0" var="ab">
      <h:column>
        <f:facet name="header">
```

```
      <h:outputText value="RoseIndia"
  styleClass="style4"/>
        </f:facet>
      </h:column>
    </h:dataTable>
  </h:panelGrid>
  <h:panelGrid width="175px"
  columns="3" border="0" cellspacing="0"
  cellpadding="0">
    <h:outputText value=" "/>
    <h:graphicImage id="gi3"
  value="images/verticle_line.gif"
  width="4" height="18"></h:graphicImage>
    <h:panelGroup>
      <h:dataTable id="dt2" border="0"
  cellpadding="0" cellspacing="0" width="250"
  var="gh">
        <h:column>
          <f:facet name="header">
            <h:outputText value="User Registration "
  styleClass="style1"/>
          </f:facet>
        </h:column>
      </h:dataTable>
    </h:panelGroup>
    <h:graphicImage id="gi4" value="images/
  horizontal_line.gif" width="25" height="4" >
    </h:graphicImage>
    <h:graphicImage id="gi8" value="images/
  horizontal_line.gif" width="5" height="4" ></
  h:graphicImage>
    <h:graphicImage id="gi6" value="images/
  horizontal_line.gif" width="260" height="4" >
    </h:graphicImage>
    <h:outputText value=" "/>
    <h:graphicImage id="gi7" value="images/
  verticle_line.gif" width="5" height="100%" >
    </h:graphicImage>
    <h:panelGroup>
      <h:dataTable id="dt3" border="0"
  cellpadding="0" cellspacing="0" width="250"
  var="gh">
        <h:column>
          <f:facet name="header">
            <h:outputText
  value="#{message.already_registered_msg}"
  style="color:red; font-weight: bold;"
  rendered="#{Bean.exist}"/>
          </f:facet>
        </h:column>
      </h:dataTable>
    </h:panelGrid width="100px" columns="2"
```

Integrating JSF, Spring and Hibernate

```
border="0" cellspacing="2" cellpadding="0">
<h:outputText value="User Name"
styleClass="style2"/>
<h:panelGroup>
<h:inputText id="userName"
value="#{Bean.userName}"
required="true" size="27" />
<f:verbatim><br/></f:verbatim>
<h:message for="userName"
styleClass="errors"/>
</h:panelGroup>
<h:outputText value="Password"
styleClass="style2"/>
<h:panelGroup>
<h:inputSecret id="Password"
value="#{Bean.pwd}"
size="27" reDisplay="true" required="true"/>
<f:verbatim><br/></f:verbatim>
<h:message for="Password"
styleClass="errors" />
</h:panelGroup>
<h:outputText value="Confirm Password"
styleClass="style2"/>
<h:panelGroup>
<h:inputSecret id="confirmPassword"
value="#{Bean.confPwd}" size="27"
reDisplay="true" required="true"/>
<f:verbatim><br/></f:verbatim>
<h:message for="confirmPassword"
styleClass="errors" />
</h:panelGroup>
<h:outputText value="Email"
styleClass="style2"/>
<h:panelGroup>
<h:inputText id="email"
value="#{Bean.email}" size="27"
required="true"/>
<f:verbatim><br/></f:verbatim>
<h:message
for="email" styleClass="errors" />
</h:panelGroup>
<h:outputText
value="Address" styleClass="style2"/>
<h:panelGroup>
<h:inputText
id="address" value="#{Bean.address}"
size="27"
required="true"/>
<f:verbatim><br/>
</f:verbatim>
<h:message
for="address" styleClass="errors" />
```

```
</h:panelGroup>
<h:outputText value="" />
<h:commandButton
value="sub" image="images/
submit_button.gif"
action="#{Bean.register}"/>
</h:panelGrid>
</h:panelGroup>
</h:panelGrid>
</h:form>
</center>
</body>
</html>
</f:view>
```

Welcome page:

If the user enters the correct information in all the fields then user is informed for the successful registration in the next page.



welcome.jsp :

This is the code for the above page. This is also a simple page in which only string for successful registration has been shown.

```
<%@ taglib uri="http://java.sun.com/jsf/
html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/
core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/
tomahawk" prefix="t"%>
<f:view>
<html>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<head>
<title>Welcome</title>
```

Integrating JSF, Spring and Hibernate

```
<link href="mycss.css" rel="stylesheet"
type="text/css"/>
</head>
<body ><center>

<h:form>
  <h:panelGrid width="100%" columns="1"
border="0" style="padding-left:10px;
padding-top:10px;"
styleClass="top_bg">
  <h:dataTable id="dt1" border="0"
cellpadding="0" cellspacing="0" var="ab">
    <h:column>
      <f:facet name="header">
        <h:outputText
value="RoseIndia" styleClass="style4"/>
      </f:facet>
    </h:column>
  </h:dataTable>
</h:panelGrid>
<h:panelGrid width="100%" columns="1"
border="0" >
  <f:verbatim>&nbsp;   </f:verbatim>
  <h:outputText value=" "/>
</h:panelGrid>
<h:outputText value="Registration is
successful." style="color:green; font-
weight:bold"/>
</h:form>
</center>
</body>
</html>
</f:view>
```

Configuration Files :

Information about message bundle, backing bean, navigation rules are to be specified in the **faces-config.xml** file. This file has been modified like below:

faces-config.xml :

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer
Faces Config 1.0//EN"
"http://java.sun.com/dtd/web-
facesconfig_1_0.dtd" >
```

```
<faces-config>
<application>
  <locale-config>
    <default-locale>en</default-locale>
  </locale-config>
  <message-
bundle>net.roseindia.web.ui.messages</
message-bundle>
</application>

<managed-bean>
  <managed-bean-name>Bean</managed-
bean-name>
  <managed-bean-
class>net.roseindia.web.ui.Bean</managed-
bean-class>
  <managed-bean-scope>session</
managed-bean-scope>
</managed-bean>

<managed-bean>
  <managed-bean-name>CheckValidUser</
managed-bean-name>
  <managed-bean-
class>net.roseindia.web.ui.CheckValidUser
</managed-bean-class>
  <managed-bean-scope>session</
managed-bean-scope>
</managed-bean>

<navigation-rule>
  <from-view-id>/pages/login.jsp</from-
view-id>
  <navigation-case>
    <from-outcome>reg</from-
outcome>
    <to-view-id>/pages/registration.jsp</
to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>success</from-
outcome>
    <to-view-id>/pages/
successLogin.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-
outcome>
    <to-view-id>/pages/login.jsp</to-
view-id>
```

Integrating JSF, Spring and Hibernate

```
</navigation-case>
</navigation-rule>

<navigation-rule>
  <from-view-id>/pages/registration.jsp</
from-view-id>
  <navigation-case>
    <from-outcome>success</from-
outcome>
    <to-view-id>/pages/welcome.jsp</
to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-
outcome>
    <to-view-id>/pages/
registration.jsp</to-view-id>
  </navigation-case>
</navigation-rule>

</faces-config>
```

web.xml :

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN"
"http://java.sun.com/dtd/web-
app_2_3.dtd">

<web-app>
<!-- Spring context Configuration Begins-->
<context-param>
  <param-name>log4jConfigLocation</
param-name>
  <param-value>/WEB-INF/
log4j.properties</param-value>
</context-param>

<context-param>
  <param-name>contextConfigLocation</
param-name>
  <param-value>/WEB-INF/
applicationContext-hibernate.xml</param-
value>
</context-param>

<servlet>
  <servlet-name>context</servlet-name>
  <servlet-class>
```

```
org.springframework.web.context.
ContextLoaderServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<!--End Spring configuration -->

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

<context-param>
  <param-
name>javax.faces.STATE_SAVING_METHOD</
param-name>
  <param-value>client</param-value>
</context-param>

<!-- Extensions Filter -->
<filter>
  <filter-name>extensionsFilter</filter-name>
<filter-class>
org.apache.myfaces.component.html.util.
ExtensionsFilter</filter-class>
  <init-param>
    <param-name>uploadMaxFileSize</
param-name>
    <param-value>100m</param-value>
    <description>Set the size limit for
uploaded files.
      Format: 10 - 10 bytes
      10k - 10 KB
      10m - 10 MB
      1g - 1 GB
    </description>
  </init-param>
  <init-param>
    <param-name>uploadThresholdSize</
param-name>
    <param-value>100k</param-value>
    <description>Set the threshold size - files
below this limit are stored in memory, files
above
      this limit are stored on disk.
      Format: 10 - 10 bytes
      10k - 10 KB
      10m - 10 MB
```

Integrating JSF, Spring and Hibernate

```
        1g - 1 GB
    </description>
</init-param>
</filter>

<filter-mapping>
    <filter-name>extensionsFilter</filter-
name>
    <url-pattern>*.jsf</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>extensionsFilter</filter-
name>
    <url-pattern>/faces/*</url-pattern>
</filter-mapping>

<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>*.jsf</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

</web-app>
```

Creation of Beans :

Bean.java : This bean has been used for registration page. This bean has properties related to all the fields in the page and setter and getter method corresponding to all the properties. exist property has been used to set true or false value to the "rendered" attribute of the outputText tag responsible for presenting the string "User is already registered". If the username is already present then the value for exist property is set to "true". dao object of HibernateSpringDAO class is used to work with the database.

```
package net.roseindia.web.ui;
```

```
import net.roseindia.web.common.*;
```

```
import net.roseindia.dao.*;
```

CheckValidUser.java

This bean has been used in the login page. All the properties of the bean are related to the fields of the login page. When the page is submitted then checkUser() method is called which checks the username and password. If both are correct then the user is sent to the next page which welcomes the user otherwise message is displayed to the user in the same login page. In this bean there is one exist property which is set to true if username or password doesn't match with the database. So this value is set to the "rendered" property of the tag responsible for displaying the string "User name or password is incorrect".

```
package net.roseindia.web.ui;
```

```
import net.roseindia.web.common.*;
```

```
import net.roseindia.dao.*;
```

```
import javax.faces.application.FacesMessage;
```

```
import javax.faces.context.FacesContext;
```

```
public class CheckValidUser{
```

```
    String userName;
```

```
    String pwd;
```

```
    boolean exist;
```

```
    public void setUserName(String
userName){
```

```
        this.userName=userName;
```

```
    }
```

```
    public void setPwd(String pwd){
```

```
        this.pwd=pwd;
```

```
    }
```

```
    public void setExist(boolean exist){
```

```
        this.exist=exist;
```

```
    }
```

```
    public String getUserName(){
```

```
        return userName;
```

```
    }
```

```
    public String getPwd(){
```

```
        return pwd;
```

```
    }
```

Integrating JSF, Spring and Hibernate

```
public boolean getExist(){
    return exist;
}
public String checkUser() throws Exception
{
    String status = "failure";

    HibernateSpringDAO dao =
(HibernateSpringDAO)
ServiceFinder.findBean("SpringHibernateDao");

    if(dao.validateUser
(getUserName(),getPwd())!=null){
        exist=false;
        status = "success";
    }
    exist=true;
    return status;
}
}
```

MessageFactory.java :

This java code is used to get the message from message bundle of specific locale. This class has been used in the Bean class to set the appropriate message for different fields when the defined condition doesn't meet.

```
package net.roseindia.web.ui;
import net.roseindia.web.common.*;

import java.util.*;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;

public class MessageFactory {
    ResourceBundle bundle;
    Locale locale;

    public MessageFactory() {
        locale =
FacesContext.getCurrentInstance().
getViewRoot().getLocale();
        bundle = ResourceBundle.getBundle
("net.roseindia.web.ui.messages", locale);
    }

    public String getMessage(String key) {
        return bundle.getString(key);
    }
}
```

```
}
```

Creation of Properties File :

This is the file containing the messages strings that are to be shown in different pages.

Registration Page

```
errorPasswordConfirm=Passwords are not
same.
errorUserId=User ID can not be less than 4
characters.
errorUserName=User Name can not be less
than 4 characters.
errorPasswordLength=Password can not be
less than 6 characters.
errorEmail=Invalid Email Address.
already_registered_msg=User is already
registered.
```

Messages.properties file of JSF

```
javax.faces.component.UIInput.REQUIRED=Cannot
be blank
```

Creating CSS :

```
body{
background-color:#fff2f2;
margin-left:0;
margin-right:0;
margin-top:0;
margin-bottom:0;
}

.top_bg{
background-image:url(../images/TOP_BG.gif);
background-repeat:repeat-x;
}

.style1 {
font-family: Verdana, Arial, Helvetica, sans-
serif;
font-weight: bold;
font-size: 12px;
}

.style2 {
font-family: Verdana, Arial, Helvetica, sans-
serif;
font-size: 10px;
font-weight: bold;
}
```

Integrating JSF, Spring and Hibernate

```
.style3 {font-size: 13px; font-family:
Verdana, Arial, Helvetica, sans-serif;}
.style4 {
font-family: Verdana, Arial, Helvetica, sans-
serif;
font-weight:bold;
color: #FF0000;
}
.errors {
font-style: italic;
color: green;
}
```

7. Business Objects of Business Logic tier

In this section we will develop the objects of business logic tier. Business logic tier refers to the mid tier of 3-tier web application architecture.

Business Objects

Business logic tier communicates both with user interface tier and the database tier. In the business logic tier all the logic are encapsulated. The business objects and business services in the application resides in the Business logic tier. The business objects contain the data and logic associated with the data. In our application there is only one business object the User.

Here is the code of **User.java** business object:

```
package net.roseindia.dao.hibernate;

import java.io.Serializable;

/** @author Hibernate CodeGenerator */
public class User implements Serializable {

    /** identifier field */
    private Integer userId;

    /** nullable persistent field */
    private String userName;

    /** nullable persistent field */
    private String userPassword;

    /** nullable persistent field */
    private String userEmail;
```

```
    /** nullable persistent field */
    private String userAddress;

    /** full constructor */
    public User(Integer userId, String
userName, String userPassword, String
userEmail,
        String userAddress) {
        this.userId = userId;
        this.userName = userName;
        this.userPassword = userPassword;
        this.userEmail = userEmail;
        this.userAddress = userAddress;
    }

    /** default constructor */
    public User() {
    }

    /** minimal constructor */
    public User(Integer userId) {
        this.userId = userId;
    }

    public Integer getUserId() {
        return this.userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getUserName() {
        return this.userName;
    }

    public void setUserName(String
userName) {
        this.userName = userName;
    }

    public String getUserPassword() {
        return this.userPassword;
    }

    public void setUserPassword(String
userPassword) {
        this.userPassword = userPassword;
    }
}
```

Integrating JSF, Spring and Hibernate

```
public String getUserEmail() {
    return this.userEmail;
}

public void setUserEmail(String userEmail)
{
    this.userEmail = userEmail;
}

public String getUserAddress() {
    return this.userAddress;
}

public void setUserAddress(String
userAddress) {
    this.userAddress = userAddress;
}
}
```

Since we are using Hibernate for persistence, the User business object provides getter and setter methods for all fields.

Business Services

Business Services contains the code to interact with the data tier. In this application we have developed a formal service layer, client can use these services interface. There is only one service in this application the **ServiceFinder**. The **ServiceFinder** service is used to get the Spring managed beans from **WebApplicationContext**.

In the next section we will learn about Integration tier of our application. Integration tier is also known as data access tier.

8. Implementing Data Access Layer with Hibernate

In this application we are using Hibernate to implement data access layer. Hibernate is an open source O/R mapping framework that handles all the persistence logic.

Hibernate supports all major database available in the market. The Hibernate Query Language is an object-oriented extension to SQL, which can be extensively used to save and retrieve

the data in the form of Java objects from database. Hibernate also supports association, inheritance, polymorphism, composition and also the Java Collection framework.

Data Access Object (DAO)

In this application we have used the DAO pattern. The DAO pattern abstracts and encapsulates all access to the data source. Our application has one DAO interface: **HibernateSpringDAO**. The implementation classes of it is **HibernateSpringDAOImpl** that contains Hibernate-specific logic to manage and persist data.

Here is the code of **HibernateSpringDAO.java** file:

```
import
org.springframework.dao.DataAccessException;
import net.roseindia.dao.hibernate.*;

public interface HibernateSpringDAO {

    /**
     * Retrieve all <code>>true</code>/
     * <code>>false</code> from the datastore.
     * @return a <code>>true</code> or
     * <code>false</code>.
     */
    public User checkUser(String strUserName)
throws
DataAccessException,java.sql.SQLException;

    /**
     * Retrieve all <code>>true</code>/
     * <code>>false</code> from the datastore.
     * @return a <code>>true</code> or
     * <code>false</code>.
     */
    public User validateUser(String
strUserName,String password) throws
DataAccessException,java.sql.SQLException;

    /**
     * Saves User object to the datastore.
     *
     */
    public void
addUser(net.roseindia.dao.hibernate.User
obj) throws DataAccessException;
```

Integrating JSF, Spring and Hibernate

```
}
```

Here is the code of **HibernateSpringDAOImpl.java** file that actually implements the logic:

```
package net.roseindia.dao;

import java.util.*;
import org.springframework.dao.DataAccessException;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import org.hibernate.criterion.*;
//Java Imports
import java.sql.*;
import javax.servlet.http.HttpSession;
//Project Imports
import net.roseindia.dao.hibernate.*;

public class HibernateSpringDAOImpl
extends HibernateDaoSupport
implements HibernateSpringDAO {

    public User checkUser(String strUserName)
throws DataAccessException,
java.sql.SQLException {
    User obj = null;
    DetachedCriteria criteria =
DetachedCriteria.forClass(User.class);
    criteria.add(Expression.eq("userName",
strUserName));

    List objs =
getHibernateTemplate().findByCriteria(criteria);
    if ((objs != null) && (objs.size() > 0)) {
        obj = (User) objs.get(0);
    }
    return obj;
}

    public User validateUser(String
strUserName,String password)
throws DataAccessException,
java.sql.SQLException {
    User obj = null;
    DetachedCriteria criteria =
DetachedCriteria.forClass(User.class);
    criteria.add(Expression.eq("userName",
strUserName));
```

```
criteria.add(Expression.eq("userPassword",
password));
    List objs =
getHibernateTemplate().findByCriteria(criteria);
    if ((objs != null) && (objs.size() > 0)) {
        obj = (User) objs.get(0);
    }
    return obj;
}
```

```
public void
addUser(net.roseindia.dao.hibernate.User
obj)
throws DataAccessException {
    getHibernateTemplate().save(obj);
}
;
```

Database Design

Our application contains only one table whose structure is as follows:

Field	Type	Null	Key	Default	Extra
userId	int(11)	NO	PRI		auto_increment
userName	varchar(20)	YES			
userPassword	varchar(11)	YES			
userEmail	varchar(30)	YES			
userAddress	varchar(30)	YES			

In the next section we will integrate all the components.

9. Integrating JSF, Spring and Hibernate

In this section we will explain you the process of Integrating Spring with JSF technology. This section gives you a brief description about Spring container (a `WebApplicationContext`), which contains all the 'business beans' present in the application.

Configuring Spring
`context(WebApplicationContext)`

What is WebApplicationContext?

The **WebApplicationContext** is an interface that extends the **ApplicationContext** interface in the **Spring** framework. This interface is used to provide the configuration for a web application. The **WebApplicationContext** is ready only while application is running, it can even be reloaded in runtime if the implementation

Integrating JSF, Spring and Hibernate

supports this.

Configuring `WebApplicationContext`

First of all it is necessary to configure the **`WebApplicationContext`** as a **`ContextListener`** in the **`web.xml`** file of the web application. Following code shows the configuration:

```
<listener>
<listener-class>
org.springframework.web.context.
ContextLoaderListener
</listener-class>
</listener>
```

In case you are using an older version of the Servlet API (<2.3), you have to use Spring's `ContextLoaderServlet` in order to configure `WebApplicationContext`. You can use the following code in the `web.xml` file:

```
<servlet>
  <servlet-name>context</servlet-name>
  <servlet-class>
    org.springframework.web.
    context.ContextLoaderServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

The `ContextLoaderListener` uses the configuration defined in `applicationContext-hibernate.xml` file and creates the object of `WebApplicationContext` for the application. The path of `applicationContext-hibernate.xml` is passed as `<context-param>..</context-param>` property. Following code can be used for this purpose:

```
<context-param>
  <param-name>contextConfigLocation</
  param-name>
  <param-value>/WEB-INF/
  applicationContext-hibernate.xml</param-
  value>
</context-param>
```

When the application is started,

`ContextLoaderListener` loads the configuration parameters from `applicationContext-hibernate.xml` file and creates an object of `WebApplicationContext` and stores it the `ServletContext` of the web application.

Now our application can use the `WebApplicationContext` to find the beans present in it.

Getting the reference of Application context:

```
ApplicationContext appContext =
WebApplicationContextUtils.
getWebApplicationContext
(servletContext);
```

Getting the bean from the Application Context

```
Object o =appContext.getBean(beanName);
```

Developing Service Finder

In our application we will use `Service Finder` class to find the beans managed by Spring framework. Here is the code of the `Service finder utility (ServiceFinder.java)`:

```
package net.roseindia.web.common;

import javax.faces.context.FacesContext;
import javax.faces.context.ExternalContext;

import javax.servlet.ServletContext;

import org.springframework.context.ApplicationContext;

import org.springframework.web.context.support.
WebApplicationContextUtils;

import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;

public class ServiceFinder {
```

Integrating JSF, Spring and Hibernate

```
public static Object findBean(String
beanName){
    FacesContext context=
FacesContext.getCurrentInstance();

    ServletContext servletContext =
    (ServletContext)context.getExternalContext().
getContext();
    ApplicationContext appContext =
    WebApplicationContextUtils.
getWebApplicationContext(servletContext);
Object o =appContext.getBean(beanName);
    return o;
}
}
```

The findBean() method of ServiceFinder class is used to get the reference of in the backing beans of JSF.

10. JSF, Integrating Presentation Layer
In this section we will learn about configuring the presentation layer.

The presentation tier integration actually involves the following steps:

1. Creating JSP pages

The JSP pages used in this application are login.jsp, registration.jsp, successLogin.jsp and welcome.jsp. These pages are already described in the previous section Developing Login and Registration form and backing beans.

2. Defining the page navigation rules

Following code present in the faces-config.xml file defines the page navigation rule.

```
<navigation-rule>
  <from-view-id>/pages/login.jsp</from-
view-id>
  <navigation-case>
    <from-outcome>reg</from-outcome>
    <to-view-id>/pages/
registration.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>success</from-
outcome>
    <to-view-id>/pages/
```

```
successLogin.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-
outcome>
    <to-view-id>/pages/login.jsp</to-
view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/pages/
registration.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-
outcome>
    <to-view-id>/pages/
welcome.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>failure</from-
outcome>
    <to-view-id>/pages/
registration.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

3. Developing and configuring backing beans

In our [application](#) there are two backing beans **Bean** and **CheckValidUser**.

Following code in the **faces-config.xml** file declares the backing beans:

```
<application>
  <locale-config>
    <default-locale>en</default-locale>
  </locale-config>
  <message-
bundle>net.roseindia.web.ui.messages
</message-bundle>
</application>

<managed-bean>
  <managed-bean-name>Bean</managed-
bean-name>
  <managed-bean-
class>net.roseindia.web.ui.Bean
</managed-bean-class>
  <managed-bean-scope>session
</managed-bean-scope>
</managed-bean>
```

Integrating JSF, Spring and Hibernate

```
<managed-bean>
  <managed-bean-
name>CheckValidUser</managed-bean-
name>
  <managed-bean-
class>net.roseindia.web.ui.CheckValidUser</
managed-bean-class>
  <managed-bean-scope>session
</managed-bean-scope>
</managed-bean>
```

4. Integrating JSF with business logic tier

We are using ServiceFinder class to get the Spring managed bean.

```
FacesContext context=
FacesContext.getCurrentInstance();

ServletContext servletContext =
(ServletContext)context.getExternalContext().
getContext();
ApplicationContext appContext =
WebApplicationContextUtils.
getWebApplicationContext(servletContext);
```

11. Integrating Business Logic Tier and Integration Tier Components

In the business logic tier web have created business objects, business services and now we are going to integrate them using Spring framework.

Business Objects

In this example we have only one business object the User.

Integration tier components

Hibernate maps the business objects to database using XML configuration file. Following file (**User.hbm.xml**) is used to map User object with the database.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//
Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
```

```
<hibernate-mapping auto-import="true"
default-lazy="false">

<class
name="net.roseindia.dao.hibernate.User"
table="users">

<id
name="userId"
type="java.lang.Integer"
column="userId">
<generator class="increment" />
</id>

<property
name="userName"
type="java.lang.String"
column="userName"
length="20"/>

<property
name="userPassword"
type="java.lang.String"
column="userPassword"
length="11"/>

<property
name="userEmail"
type="java.lang.String"
column="userEmail"
length="30" />

<property
name="userAddress"
type="java.lang.String"
column="userAddress"
length="30"/>

<!-- Associations -->
</class>
</hibernate-mapping>
```

The **HibernateSpringDAO** is wired with HibernateTemplate by Spring:

```
<bean id="HibernateSpringDaoTarget"
class="net.roseindia.dao.HibernateSpringDAOImpl">
<property name="sessionFactory"><ref
local="sessionFactory"/></property>
</bean>
```

Integrating JSF, Spring and Hibernate

Here is the full code of **applicationContext-hibernate.xml** file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD
BEAN//EN" "http://
www.springframework.org/dtd/spring-
beans.dtd">

<!--
- Application context definition for MyFaces,
Hibernate and Spring Integration application.
-->
<beans>

<!-- =====
RESOURCE DEFINITIONS
===== -->

<!-- Configurer that replaces ${...}
placeholders with values from a properties file
-->
<!-- (in this case, JDBC-related settings for
the dataSource definition below) -->
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.
PropertyPlaceholderConfigurer">
<property name="location"><value>/WEB-
INF/jdbc.properties</value></property>
</bean>

<!-- Local DataSource that works in any
environment -->
<!-- Note that DriverManagerDataSource
does not pool; it is not intended for
production -->

<bean id="dataSource"
class="org.springframework.jdbc.datasource.
DriverManagerDataSource">
<property name="driverClassName">
<value>${jdbc.driverClassName}</value>
</property>
<property name="url"><value>${jdbc.url}</value></property>
<property
name="username"><value>${jdbc.username}</value></property>
<property
name="password"><value>${jdbc.password}</value></property>
```

```
</bean>

<!-- JNDI DataSource for J2EE environments
-->
<!--
<bean id="dataSource"
class="org.springframework.jndi.
JndiObjectFactoryBean">
<property
name="jndiName"><value>java:comp/env/
jdbc/roseindiaDB_local</value></property>
</bean>
-->
<!-- Hibernate SessionFactory -->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.
LocalSessionFactoryBean">
<property name="dataSource">
<ref local="dataSource"/></property>
<property name="mappingResources">
<list>
<value>/net/roseindia/dao/hibernate/
User.hbm.xml</value>
</list>
</property>
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">
${hibernate.dialect}
</prop>
<prop key="hibernate.show_sql">true
</prop>
</props>
</property>
</bean>

<!-- Transaction manager for a single
Hibernate SessionFactory (alternative to JTA)
-->
<bean id="transactionManager"
class="org.springframework.orm.hibernate3.
HibernateTransactionManager">
<property name="sessionFactory"><ref
local="sessionFactory"/></property>
</bean>

<!-- =====
BUSINESS OBJECT DEFINITIONS
===== -->

<!--
Data access object: Hibernate
```

Integrating JSF, Spring and Hibernate

implementation.

—>

```
<bean id="HibernateSpringDaoTarget"
class="net.roseindia.dao.
HibernateSpringDAOImpl">
<property name="sessionFactory"><ref
local="sessionFactory"/></property>
</bean>
```

<!--

- Transactional proxy for Application's central data access object.

-

- Defines specific transaction attributes with "readOnly" markers, which is an optimization that is particularly valuable with Hibernate (to suppress unnecessary flush attempts for read-only operations).

- Note that in a real-life app with multiple transaction proxies,

- you will probably want to use parent and child bean definitions as described in the manual, to reduce duplication.

—>

```
<bean id="SpringHibernateDao"
class="org.springframework.transaction.
interceptor.TransactionProxyFactoryBean">
<property name="transactionManager">
<ref local="transactionManager"/>
</property>
<property name="target"><ref
local="HibernateSpringDaoTarget"/>
</property>
<property name="transactionAttributes">
<props>
<prop
key="get*">PROPAGATION_REQUIRED,readOnly
</prop>
<prop key="find*">
PROPAGATION_REQUIRED,readOnly
</prop>
<prop key="load*">
PROPAGATION_REQUIRED,readOnly</prop>
<prop key="store*">
PROPAGATION_REQUIRED
</prop>
<prop key="add*">
PROPAGATION_REQUIRED
</prop>
```

```
</props>
</property>
</bean>

</beans>
```

This application is supported with full free code. In the next section we will show you how you can download and install it on your computer.

12. Download full code of JSF, Spring and Hibernate based Registration program

From this page you can download the source code of the application.

Downloading code

Download the code of the application.

Extracting and Installing the code on Tomcat
Extract the downloaded file and copy **HibernateMyfaces** directory to tomcat **webapps** directory. This will install application on tomcat server.

Creating database

Follow the steps given in the previous section and create database for testing the application.

Changing the database login information

Go to **HibernateMyfaces\WEB-INF** directory and edit **jdbc.properties** to point to your database.

Compile the application

To compile the program **ant tool** must be installed on your system. Open console and go to **HibernateMyfaces\WEB-INF\src** directory and type **ant** command. This will compile your program.

Running and testing the program

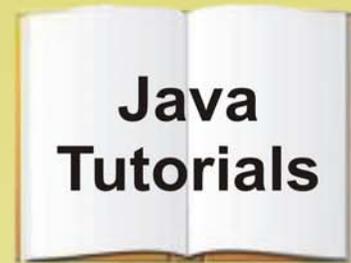
Start tomcat and type **http://localhost:8080/HibernateMyfaces/**. You browser will display the login page as shown below.

Integrating JSF, Spring and Hibernate



Now you can test the application.

Congratulations you have successfully developed application using JSF, Hibernate and Spring frameworks.



 **RoseIndia**
<http://www.roseindia.net>

Facelet

Facelet is a view technology for Java Server Faces (JSF) that allows building composite views more quickly and easily than with JSP which is the default view technology for JSF. JSP pages are compiled into servlets but it's not the case with Facelets because Facelet pages are XML compliant and its framework uses a fast SAX-based compiler to build views. Facelets can make changes to pages immediately so developing JSF applications with Facelets is simply faster. This section explains all facelet tags. Download zip file from the link given below of the page and run it. You will get the page given below and can see the output of each tag clicking the related link.



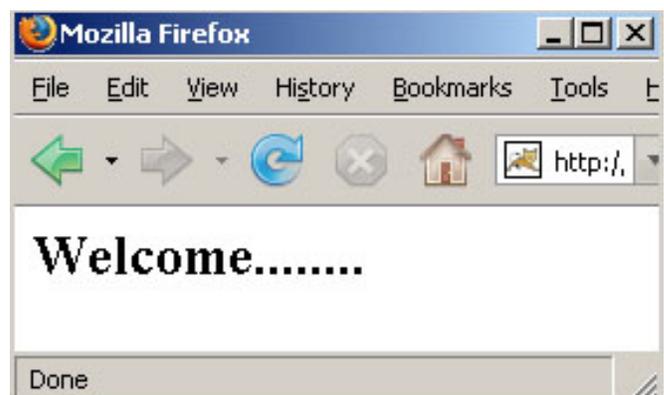
1.Facelet component tag

This tag is used to add a new component into the JSF component tree as children of UI component instance. This tag shows its behavior like composition tag. The difference is that the component tag inserts a new UIcomponent instance in the component tree and this instance is the root of all its child components or fragments. The content outside of the tag is ignored as it happens with composition tag.

comptemplate.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <head>
    <title>Facelet component tag </title>
  </head>
  <body>
    Content above component tag will not be rendered.
    <ui:component >
      <h2>Welcome.....</h2>
    </ui:component >
    Content below component tag will not be rendered.
  </body>
</html>
```

Rendered Output:



Facelet

2. Facelet composition tag

This is a templating tag and is used for the wrapping the content that can be included in any other facelet. This tag provides some useful features. Any content outside of this tag is left be rendered. You can include normal html content in your page but Facelet will render only content that is within this tag i.e. composition tag. This tag takes one attribute named "template". This attribute is set to the path of the template where the content of this tag will be included.

In the code below we have taken template attribute, which indicates the template to which the content inside this composition tag will be rendered. Content outside of the composition tag will not be rendered. In the comtemplate.xhtml we have used insert tag to include the content inside the composition tag to the comtemplate.xhtml page.

composition.xhtml:

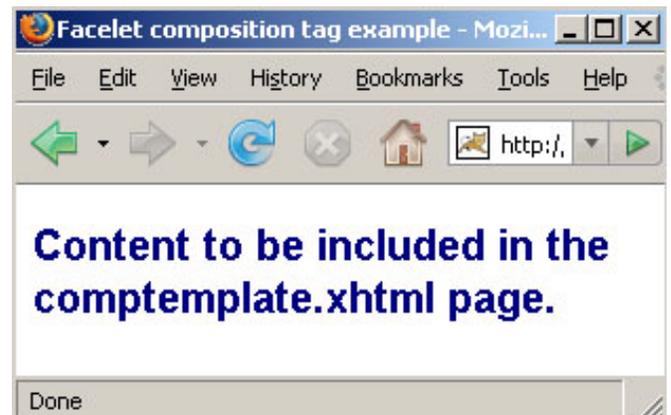
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets">
  <body>
    Content above composition tag will not be rendered.
    <ui:composition template="/pages/composition/comtemplate.xhtml">
      <h2>Content to be included in the comtemplate.xhtml page.</h2>
    </ui:composition>
    Content below composition tag will not be rendered.
  </body>
</html>
```

comtemplate.xhtml :

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

```
xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <head>
    <title>Facelet composition tag example</title>
    <link href="../../style/CSS.css"
rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <ui:insert />
  </body>
</html>
```

Rendered Output:



3. Facelet debug tag

This tag is useful in displaying the component tree and scoped variables. This information will be displayed in a popup window of browser when we press Ctrl+Shift+(a key). This key will be specified in hotkey attribute. For example, in the code below in "debugtemplate.xhtml", this has been specified "p". So when page comes to the user then if Ctrl+Shift+p is pressed, debug window is open which displays the component tree and scoped variables.

debug.xhtml :

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

Facelet

```
xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
<body>
  Content above composition tag will not be
  rendered.
  <ui:composition template="/pages/debug/
  debugtemplate.xhtml">
    <ui:define name="face1">
      <h2>Java Jazz Up</h2>
      <h3>Facelet Examples</h3>
    </ui:define>
    <ui:define name="face2">Enter UserID
  :<br/>
    <h:inputText id="it" /><br/><br/>
    </ui:define>
    <ui:define name="face3">Enter Password
  :<br/>
    <h:inputSecret id="is" /><br/><br/>
    </ui:define>
    <ui:define name="face4">
      <h:commandButton id="cb"
  value="Submit" />
    </ui:define>
  </ui:composition>
  Content below composition tag will not be
  rendered.
</body>
</html>
```

debugtemplate.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
  xmlns:ui="http://java.sun.com/jsf/
  facelets">
  <head>
    <title>Facelet debug tag example</title>
    <link href="../style/CSS.css"
  rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <ui:insert name="face1"></ui:insert>
    <ui:insert name="face2"></ui:insert>
    <ui:insert name="face3"></ui:insert>
    <ui:insert name="face4"></ui:insert>
    <ui:debug hotkey="p" rendered="true"/>
```

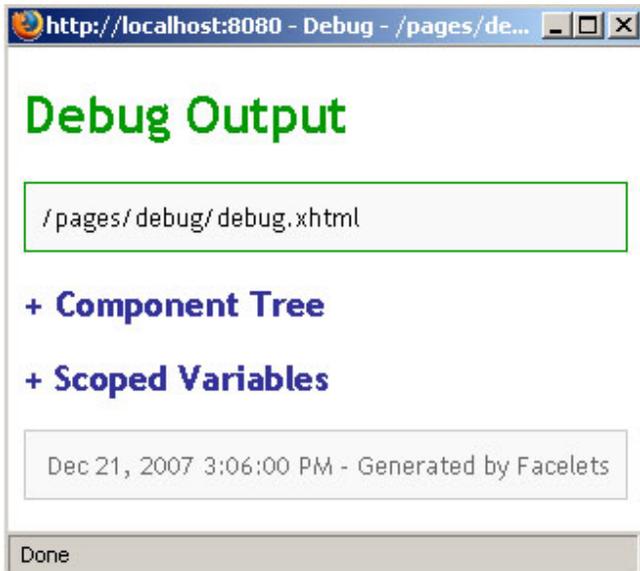
```
</body>
</html>
```

Rendered Output: This is the page that is displayed to the user first. Now if we press Ctrl+Shift+p then debug window is opened that is shown in the second figure below:



Facelet

Debug window:



4. Facelet decorate tag

This tag is like composition tag. Difference between those is that the content outside of the decorate tag is rendered while it is reverse for composition tag i.e. it is not rendered when we use composition tag. This tag is useful when we want content with some decoration text in the document.

decorate.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
<body>
<h2>Java Jazz Up</h2>
<h3>Facelet Examples</h3>
<hr/>
<ui:decorate template="/pages/decorate/
decoratetemplate.xhtml">
<ui:define name="face1">
<table border="1">
<tr><th>User</th>
<th>Email</th></tr>
<tr><td>ABC</td>
```

```
<td>abc@javajazzup.com</td></tr>
<tr><td>XYZ</td>
<td>xyz@javajazzup.com</td></tr>
</table><hr/>
</ui:define>
</ui:decorate>
<h3>Content below decorate tag.</h3>
</body>
</html>
```

decoratetemplate.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
xmlns:ui="http://java.sun.com/jsf/
facelets">
<head>
<title>Facelet decorate tag example</title>
<link href="../../style/CSS.css"
rel="stylesheet" type="text/css"/>
</head>
<body>
<ui:insert name="face1"></ui:insert>
</body>
</html>
```

Facelet

Rendered Output:



5. Facelet define tag

This tag is used to define the name of the content. This named content can be included within a template. This tag is used within those tags that allows templating like composition and decorate tags. This tag takes one attribute named "name" that is required to be included when using this define tag. This name attribute is required to be same as name attribute of insert tag in the target template to include the content specified in define tag with the same name. For example, in the first define tag name attribute is set to "face1". Now look at the code below in "definetemplate.xhtml" where we have used insert tag with name attribute. This name attribute is given value "face1". So the content within define tag, whose name attribute value matches with the name attribute of the insert tag i.e."face1", will be included in the "definetemplate.xhtml".

define.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
  <body>
    Content above composition tag will not be
    rendered.
    <ui:composition template="/pages/define/
    definetemplate.xhtml">
      <ui:define name="face1">
        <h2>Java Jazz Up</h2>
        <h3>Facelet Examples</h3>
      </ui:define>
      <ui:define name="face2">Enter UserID
      :<br/>
        <h:inputText id="it" /> <br/> <br/>
      </ui:define>
      <ui:define name="face3">Enter Password
      :<br/>
        <h:inputSecret id="is" /> <br/> <br/>
      </ui:define>
      <ui:define name="face4">
        <h:commandButton id="cb"
        value="Submit" />
      </ui:define>
    </ui:composition>
    Content below composition tag will not be
    rendered.
  </body>
</html>
```

definetemplate.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
xmlns:ui="http://java.sun.com/jsf/
facelets">
  <head>
    <title>Facelet define tag example</title>
    <link href="../../style/CSS.css"
    rel="stylesheet" type="text/css"/>
```

Facelet

```
</head>
<body>
<ui:insert name="face1"> </ui:insert>
<ui:insert name="face2"> </ui:insert>
<ui:insert name="face3"> </ui:insert>
<ui:insert name="face4"> </ui:insert>
</body>
</html>
```

Rendered Output:



6. Facelet fragment tag

This tag is used to insert the new UIComponent to the component tree and the content outside of the tag is included to the tree. So this tag is same with component tag as decorate tag is with composition tag i.e. as decorate tag behaves same as composition tag except including content outside the tag, in the same way fragment tag behaves same as component tag except including content

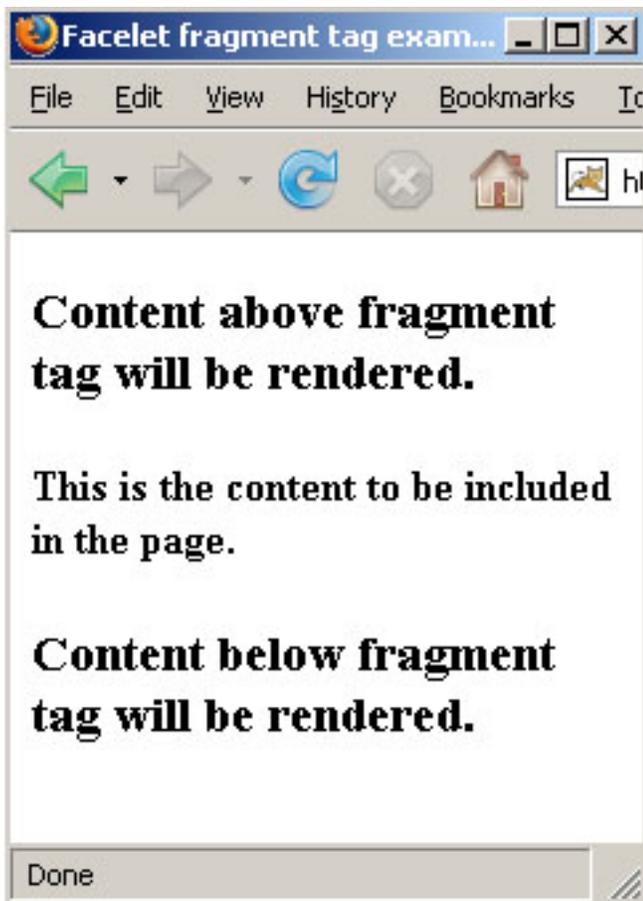
outside the tag. In this example, the content within fragment tag is included in component tree and the code above fragment tag is also rendered. So "Content above fragment tag will be rendered." and "Content below fragment tag will be rendered." is rendered.

fragmenttemplate.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <head>
    <title>Facelet fragment tag example</title>
  </head>
  <body>
    <h3>Content above fragment tag will be rendered.</h3>
    <ui:fragment >
      <h4>This is the content to be included in the page.</h4>
    </ui:fragment >
    <h3>Content below fragment tag will be rendered.</h3>
  </body>
</html>
```

Rendered Output:

Facelet



7. Facelet include tag

This tag is used to include the content of a page. This page name is specified by src attribute of include tag. The page that has been included should use composition tag or component tag. It may contain xhtml or xml to be included. In the program below, we have used include tag and src attribute is set to "includepage.xhtml". So the content of this page will be included in the "include.xhtml" page and rendered in "includetemplate.xhtml" because insert tag with name attribute set to face5 is used in "includetemplate.xhtml".

include.xhtml:

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

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
<body>
  Content above composition tag will not be rendered.
  <ui:composition template="/pages/include/includetemplate.xhtml">
    <ui:define name="face1">
      <h2>Java Jazz Up</h2>
      <h3>Facelet Examples</h3>
    </ui:define>
    <ui:define name="face2">Enter UserID
  :<br/>
    <h:inputText id="it" /> <br/> <br/>
    </ui:define>
    <ui:define name="face3">Enter Password
  :<br/>
    <h:inputSecret id="is" /> <br/> <br/>
    </ui:define>
    <ui:define name="face4">
      <h:commandButton id="cb"
value="Submit" />
    </ui:define>
    <ui:define name="face5">
      <ui:include src="includepage.xhtml"/>
    </ui:define>
    </ui:composition>
    Content below composition tag will not be rendered.
  </body>
</html>
```

includetemplate.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
  <title>Facelet include tag example</title>
  <link href="../../style/CSS.css"
rel="stylesheet" type="text/css"/>
</head>
<body>
  <ui:insert name="face1"> </ui:insert>
```

Facelet

```
<ui:insert name="face2"> </ui:insert>
<ui:insert name="face3"> </ui:insert>
<ui:insert name="face4"> </ui:insert>
<ui:insert name="face5"></ui:insert>
</body>
</html>
```

includepage.xhtml :

The content in this page will be included in the **"includetemplate.xhtml"**.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
    xmlns:ui="http://java.sun.com/jsf/
facelets">

    <ui:composition>
    <br/><br/>This is the content of
<b>"includepage.xhtml"</b>
    </ui:composition>

</html>
```

Rendered Output:



8. Facelet insert tag

This tag is used to replace the content defined in another facelet to the template. This tag takes one attribute that is not a required attribute and is used in conjunction with define tag. If you set this attribute same as defined in define tag then that content within define tag will be included. If it doesn't match then the content specified within opening and closing tag of this insert tag will be displayed. For example, in the code below in "insert.xhtml" there is not any define tag whose name attribute value is "face5" and this value is used in the second file "inserttemplate.xhtml". So the content ("This is the default text rendered") specified within opening and closing tag of insert tag is displayed. While there is one insert tag whose value of name attribute ("face1") matches with that of define tag, so the content " Java Jazz Up " and " Facelet Examples" will be replaced to the insert tag.

insert.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html">
    <body>
        Content above composition tag will not be
        rendered.
        <ui:composition template="/pages/insert/
inserttemplate.xhtml">
            <ui:define name="face1">
                <h2>Java Jazz Up</h2>
                <h3>Facelet Examples</h3>
            </ui:define>
            <ui:define name="face2">Enter UserID
: <br/>
                <h:inputText id="it" /> <br/> <br/>
            </ui:define>
            <ui:define name="face3">Enter Password
: <br/>
                <h:inputSecret id="is" /> <br/> <br/>
            </ui:define>
            <ui:define name="face4">
                <h:commandButton id="cb"
value="Submit" />
```

Facelet

```
</ui:define>
</ui:composition>
Content below composition tag will not be
rendered.
</body>
</html>
```

inserttemplate.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
xmlns:ui="http://java.sun.com/jsf/
facelets">
<head>
<title>Facelet insert tag example</title>
<link href="../../style/CSS.css"
rel="stylesheet" type="text/css"/>
</head>
<body>
<ui:insert name="face1"></ui:insert>
<ui:insert name="face2"> </ui:insert>
<ui:insert name="face3"> </ui:insert>
<ui:insert name="face4"> </ui:insert>
<ui:insert name="face5"><h3>This is the
default text rendered</h3> </ui:insert>
</body>
</html>
```

Rendered Output:



9. Facelet param tag

This tag is used to pass objects as variables between facelets. This tag has two required attributes name and value. name attribute is the name of the variable and the value attribute is to set the value of this variable. You can use this tag where a define tag is used within composition or decorate tag. We can also use this tag within include tag. In this example, we have taken two variables user and pwd within include tag in param.xhtml and values are set through bean's properties userid and password. These variable are passed to the "includeparampage.xhtml" where we can use these variables in this line of code "Your #{user} and #{pwd} will not be disclosed".

Facelet

param.xhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
  <body>
    Content above composition tag will not be
    rendered.
    <ui:composition template="/pages/param/
paramtemplate.xhtml">
      <ui:define name="face1">
        <h2>Java Jazz Up</h2>
        <h3>Facelet Examples</h3>
      </ui:define>
      <ui:define name="face2">Enter UserID
: <br/>
        <h:inputText id="it" /><br/><br/>
      </ui:define>
      <ui:define name="face3">Enter Password
: <br/>
        <h:inputSecret id="is" /><br/><br/>
      </ui:define>
      <ui:define name="face4">
        <h:commandButton id="cb"
value="Submit" />
      </ui:define>
      <ui:define name="face5">
        <ui:include
src="includeparampage.xhtml">
          <ui:param name="user"
value="#{MessageBean.userid}"/>
          <ui:param name="pwd"
value="#{MessageBean.password}"/>
        </ui:include>
      </ui:define>
    </ui:composition>
    Content below composition tag will not be
    rendered.
  </body>
</html>
```

paramtemplate.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
xmlns:ui="http://java.sun.com/jsf/
facelets">
  <head>
    <title>Facelet param tag example</title>
    <link href="../../style/CSS.css"
rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <ui:insert name="face1"></ui:insert>
    <ui:insert name="face2"></ui:insert>
    <ui:insert name="face3"></ui:insert>
    <ui:insert name="face4"></ui:insert>
    <ui:insert name="face5"></ui:insert>
  </body>
</html>
```

includeparampage.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/
xhtml"
xmlns:ui="http://java.sun.com/jsf/
facelets">
  <ui:composition>
    <br/><br/><h3>Your #{user} and
#{pwd} will not be disclosed.</h3>
  </ui:composition>
</html>
```

Facelet

Rendered Output:



10. Facelet remove tag

This tag is used to remove content within this tag from a facelet at compile time. This tag don't have any attribute. This tag can be used with jsfc attribute which shows that the particular tag will be removed. In this example, the line "This line will be removed" will be removed from facelet at the time of compilation and so will not be displayed. In this example, the line where jsfc attribute is used and set to the ui:remove is not considered for compilation so input text box for this line of code will not be displayed.

remove.xhtml:

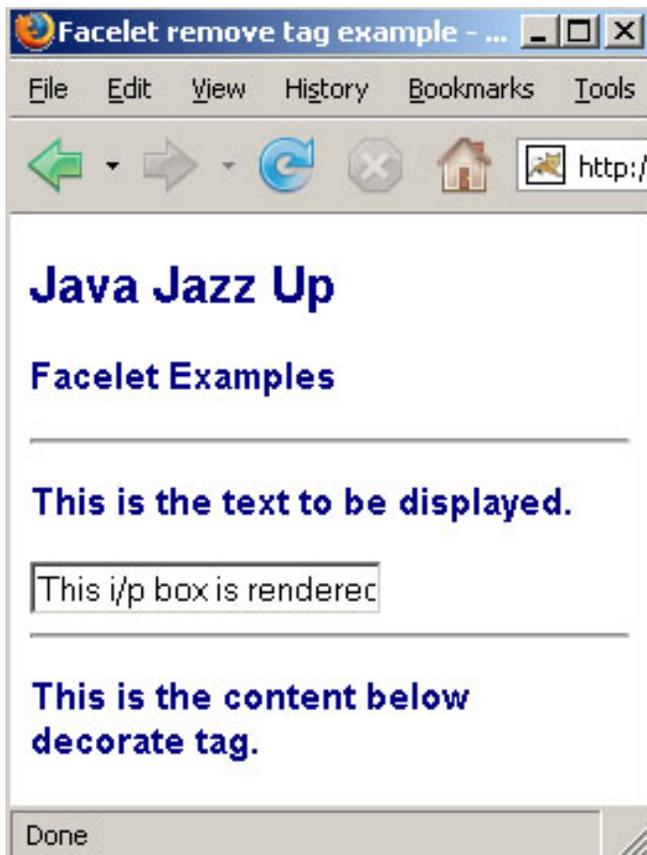
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
  <body>
    <h2>Java Jazz Up</h2>
    <h3>Facelet Examples</h3>
    <hr/>
    <ui:decorate template="/pages/remove/removetemplate.xhtml">
      <ui:define name="face1">
        <h3><h:outputText value="This is the text to be displayed."/></h3>
        <input type="text" jsfc="h:inputText" value="This i/p box is rendered" />
        <input type="text" jsfc="ui:remove" value="IT" />
        <ui:remove>This line will be removed</ui:remove>
      </ui:define>
    </ui:decorate>
    <hr/><h3>This is the content below decorate tag.</h3>
  </body>
</html>
```

removetemplate.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets">
  <head>
    <title>Facelet remove tag example</title>
    <link href="../style/CSS.css" rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <ui:insert name="face1"></ui:insert>
  </body>
</html>
```

Facelet

Rendered Output:



11. Facelet repeat tag:

This tag is used to iterate over the list of items. The name of list of items is specified by the EL expression in the value attribute of this tag. This tag contains two attributes "value" "name". The literal name specified in the name attribute is used to iterate over the items. In this example, we have used a bean named "TableBean" and info name is given to be used further. For ex., info.id, info.name used in value attribute of inputText JSF tag where id and name are attributes specified in bean. so here all id and names will be displayed.

repeat.xhtml:

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

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
<body>
  <h2>Java Jazz Up</h2>
  <h3>Facelet Examples</h3>
  <hr/>
  <ui:decorate template="/pages/repeat/repeattemplate.xhtml">
    <ui:define name="face1">
      <h3><h:outputText value="This is the list of ID and Names."/></h3>
      <ui:repeat
value="#{TableBean.perInfoAll}" var="info">
        <li>
          <h:inputText value="#{info.id}" />
          <h:inputText value="#{info.name}" />
        </li>
      </ui:repeat>
    </ui:define>
  </ui:decorate>
</body>
</html>
```

repeattemplate.xhtml :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
  <title>Facelet repeat tag example</title>
  <link href="../../style/CSS.css"
rel="stylesheet" type="text/css"/>
</head>
<body>
  <ui:insert name="face1"></ui:insert>
</body>
</html>
```

TableBean.java :(Java Bean used for collection of items)

```
package javajazzup;
public class TableBean {
  private perInfo[] perInfoAll = new
perInfo[] {
```

Facelet

```
        new perInfo(101, "Emp1",
"9891444444", "aaa", 11111),
        new perInfo(102, "Emp2",
"9911666666", "bbb", 22222),
        new perInfo(103, "Emp3",
"9313888888", "ccc", 33333)
    };
    public perInfo[] getperInfoAll() {
        return perInfoAll;
    }
}

public class perInfo {
    int id;
    String name;
    String phone;
    String city;
    int pin;
    public perInfo(int id, String name, String
phone, String city, int pin) {
        this.id = id;
        this.name = name;
        this.phone = phone;
        this.city = city;
        this.pin= pin;
    }
    public int getid() {
        return id;
    }
    public String getname() {
        return name;
    }
    public String getphone() {
        return phone;
    }
    public String getcity() {
        return city;
    }
    public int getpin() {
        return pin;
    }
}
}
```

Rendered Output:



Design Patterns

Behavioral Patterns Behavioral patterns are those patterns, which are specifically concerned with communication (interaction) between the objects. The interactions between the objects should be such that they are talking to each other and are still loosely coupled. The loose coupling is the key to n-tier architectures. In this, the implementations and the client should be loosely coupled in order to avoid hard coding and dependencies. The behavioral patterns are:

1. **Chain of Responsibility Pattern**
2. **Command Pattern**
3. **Interpreter Pattern**
4. **Iterator Pattern**
5. **Mediator Pattern**
6. **Memento Pattern**
7. **Observer Pattern**
8. **State Pattern**
9. **Strategy Pattern**
10. **Template Pattern**
11. **Visitor Pattern**

In this Issue, we are going to discuss only the Iterator, Mediator, and the Memento design patterns.

Iterator Pattern

Iterator pattern is the mechanism of accessing all the objects in a collection. To sequentially access the objects of a collection, the iterator pattern defines an interface containing the methods that access the objects of a collection or the aggregate objects without having knowledge about its internal representation. It provides uniform interface to traverse collections of all kinds.

Aggregate objects are the objects containing the group of objects as a unit. We can also referred them as container or collection. Hash table and linked list are the examples of aggregate objects.

Example: Lets take an example of employees of a software company and their section, then we add some enumeration capabilities to the Employee class. This class is the collection of employees having their names; employee id and their department and these employees are

stored in a Vector.

Now we simply the enumeration of the Vector itself just to obtain the enumeration of all the employees of the collection.

Filtered Enumeration: Suppose we want the employees of the development section. This requires a special enumeration class that access the employees belong only to the development department. The element() method we have defined provides filtered access. Now Enumeration that only returns employees related to the development section is required.

Person.java

```
import java.util.*;
interface Person {
    public abstract double income();
}
```

Teacher.java

```
class Teacher implements Person {
    private double MonthlyIncome;
    private String name;
    public Teacher(String name, double i) {
        this.name = name;
        setMonthlyIncome(i);
    }

    void setMonthlyIncome(double i) {
        if (i > 0) {
            MonthlyIncome = i;
        } else
            MonthlyIncome = 0;
    }

    public double income() {
        return MonthlyIncome;
    }
    public String getName() {
        return name;
    }
    public String toString() {
        return "Teacher: " + getName();
    }
}
```

Design Patterns

Doctor.java

```
class Doctor implements Person {
    private double feeperPatient;
    private int NoOfPatient;
    private String name;
    public Doctor(String name, double f, int n) {
        this.name = name;
        setFeeperPatient(f);
        setNoOfPatient(n);
    }

    void setFeeperPatient(double f) {
        if (f > 0)
            feeperPatient = f;
        else
            feeperPatient = 0;
    }

    void setNoOfPatient(int n) {
        if ( n > 0)
            NoOfPatient = n;
        else
            NoOfPatient = 0;
    }
    public String getName() {
        return name;
    }
    public double income() {
        return NoOfPatient * feeperPatient;
    }

    public String toString() {
        return "Doctor : " + getName();
    }
}
```

PersonTest.java

```
import java.util.Iterator;
import java.util.ArrayList;
import java.util.List;

class PersonTest {
    public static void main(String[] args) {
        List list = new ArrayList();
        list.add(new Teacher("Bill", 800.00));
        list.add(new Doctor("Al", 2.5, 200));
        list.add(new Teacher("Peter", 1200.00));
        list.add(new Doctor("Mark", 4.5, 333));
    }
}
```

```
System.out.println("Use built-in iterator:");
Iterator iterator = list.iterator();
while(iterator.hasNext()) {
    Person pr = (Person)iterator.next();
    if (pr instanceof Teacher) {
        System.out.print("Teacher has " + pr + "
income $");
        System.out.println(pr.income());
    }
}
}
```

The above example also shows a dynamic binding feature, which is popular in Object-Oriented realm.

If you want to pick up a specific object from the aggregated list, you may use the following code.

II. Mediator Pattern

Mediator pattern takes an object that puts the logic to manages state changes of other objects rather than distributing the logic among the various objects that results in decreasing the coupling between the other objects.

Define an object that encapsulates how a set of objects interacts. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and lets you vary their interaction independently.

Lets try to under stand more clearly, When we starts developing an application that have few classes and these classes interact with each other just to produce a result. As soon as the application becomes large then the logic becomes more complex and functionality increases. In such condition it is difficult to maintain this code then Mediator pattern solves the problem by maintaining the code. It loose-couples to the classes so that only one class (Mediator) has the information about rest of the classes, rest of the classes only interacts with the Mediator.

Design Patterns

Mediator.java

```
import java.io.*;

abstract class Mediator{
public abstract void pilotChanged(Pilot
theChangedPilot);

public static void main(String args[]){

ConcreteMediator aConcreteMediator = new
ConcreteMediator();
aConcreteMediator.createConcreteMediator();
(aConcreteMediator.getOnePilot()).Show();
(aConcreteMediator.getNextPilot()).Show();
OnePilot newPilot = new
OnePilot(aConcreteMediator, "OnePilot");
aConcreteMediator.pilotChanged(
(OnePilot)newPilot );
(aConcreteMediator.getOnePilot()).Show();
(aConcreteMediator.getNextPilot()).Show();
}
}
```

Pilot.java

```
abstract class Pilot{

private Mediator mediator;

public Pilot(Mediator m){
mediator = m;
}

public void changed(){
mediator.pilotChanged(this);
}

public Mediator getMediator(){
return(mediator);
}
public abstract void Show();
}
```

ConcreteMediator.java

```
class ConcreteMediator extends Mediator{

private OnePilot aOnePilot;
private NextPilot aNextPilot;

public void pilotChanged( Pilot
```

```
theChangedPilot ){

if (((
OnePilot)theChangedPilot).getSelection().equals(
aOnePilot.getSelection() ) ){
aNextPilot.setText( aOnePilot.getSelection() );
}
}

public void createConcreteMediator(){
aOnePilot = new OnePilot(this, "OnePilot");
aNextPilot = new NextPilot(this, "NextPilot");
}

public OnePilot getOnePilot(){
return(aOnePilot);
}

public NextPilot getNextPilot(){
return(aNextPilot);
}
}
```

OnePilot.java

```
class OnePilot extends Pilot{

private String text;
public OnePilot(Mediator m, String s){

super( m );
text = s;
}

public void setText(String txt){
text = txt;
}

public String getSelection(){
return(text);
}

public void Show(){
System.out.println("OnePilot = " + text);
}
}
```

NextPilot.java

Design Patterns

```
class NextPilot extends Pilot{

private String text;
public NextPilot(Mediator m, String s){

super( m );
text = s;
}

public void setText(String txt){
text = txt;
}

public String getSelection(){
return( text );
}

public void Show(){
System.out.println("NextPilot = " + text);
}
}
```

III. Memento

The memento design pattern is that pattern in which one object stores the previous state (undo via rollback) of another object. This pattern operates on a single object.

The objects originator and the caretaker use this design pattern. The object originator maintains the original state while the object caretaker assist to the originator. First the caretaker object asked to the originator for the memento object then it does the operation (or sequence of operations) it is going to do. It returns the memento object to the originator to roll back the state before the operations. Be careful while using this pattern, as the originator may change other objects or resources.

To develop a memento featured program we need to combine the design patterns like Iterator, Mediator or Command.

Here we are taking an example that uses the Mediator design pattern.

To demonstrate the concept of Memento design pattern we are taking an example of dice that

records the dice numbers.

Command.java

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.awt.FontMetrics;
```

```
interface Command {
void execute();
}
```

Memento.java

```
class Memento {
int number;
Memento(int num) {
number = num;
}
int getNumber() {
return number;
}
}
```

ButtonDice.java

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.awt.FontMetrics;
```

```
class ButtonDice extends JButton implements
Command {
Mediator mediator;
ButtonDice(ActionListener al, Mediator med) {
super("Throw Dice");
addActionListener(al);
mediator = med;
mediator.registerDice(this);
}
public void execute() {
mediator.throwit();
}
}
```

ButtonClear.java

Design Patterns

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.awt.FontMetrics;

class ButtonClear extends JButton
implements Command {
    Mediator mediator;
    ButtonClear(ActionListener al, Mediator med)
    {
        super("Clear");
        addActionListener(al);
        mediator = med;
        mediator.registerClear(this);
    }
    public void execute() {
        mediator.clear();
    }
}
```

ButtonPrevious.java

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.awt.FontMetrics;

class ButtonPrevious extends JButton
implements Command {
    Mediator mediator;
    ButtonPrevious(ActionListener al, Mediator
med) {
        super("Previous");
        addActionListener(al);
        mediator = med;
        mediator.registerPrevious(this);
    }
    public void execute() {
        mediator.previous();
    }
}
```

Display.java

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.awt.FontMetrics;

class Display extends JLabel{
    Mediator mediator;
    Display (Mediator med) {
        super("0",JLabel.CENTER);
        mediator = med;
        mediator.registerDisplay(this);
        setBackground(Color.white);
        setBorder(new EtchedBorder(Color.blue,
Color.green));
        Font font = new Font("Arial",Font.BOLD,40);
        setFont(font);
    }
}
```

Mediator.java

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.awt.FontMetrics;

class Mediator {
    ButtonDice btnDc;
    ButtonPrevious btnPrvs;
    ButtonClear btnClr;
    Display display;
    java.util.List list, undo;
    boolean restart = true;
    int counter = 0, ct = 0;
    //....
    Mediator() {
        list = new ArrayList();
        undo = new ArrayList();
    }
    void registerDice(ButtonDice bd) {
        btnDc = bd;
    }
    void registerClear(ButtonClear bc) {
        btnClr = bc;
    }
    void registerPrevious(ButtonPrevious bp) {
        btnPrvs = bp;
    }
}
```

Design Patterns

```
}
void registerDisplay(Display disp) {
    display = disp;
}
void throwit() {
    display.setForeground(Color.black);
    int num = (int)(Math.random()*6 +1);
    int i = counter++;
    list.add(i, new Integer(num));
    undo.add(i, new Memento(num));
    display.setText(""+num);
}

void previous() {
    display.setForeground(Color.red);
    btnDc.setEnabled(false);
    if (undo.size() > 0) {
        ct = undo.size()-1;
        Memento num = (Memento)undo.get(ct);
        display.setText(""+num.getNumber());
        undo.remove(ct);
    }
    if (undo.size() == 0)
        display.setText("0");
}
void clear() {
    list = new ArrayList();
    undo = new ArrayList();
    counter = 0;
    display.setText("0");
    btnDc.setEnabled(true);
}
}
```

MementoDemoDp.java

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.awt.FontMetrics;

class MementoDemoDp extends JFrame
implements ActionListener {
    Mediator mediator = new Mediator();
    MementoDemoDp() {
        JPanel panel = new JPanel();
        panel.add(new ButtonDice(this,mediator));
        panel.add(new
        ButtonPrevious(this,mediator));
        panel.add(new ButtonClear(this,mediator));
```

```
JPanel dice = new JPanel();
Display disp = new Display(mediator);
dice.add(disp);
getContentPane().add(dice, "Center");
getContentPane().add(panel, "South");
setTitle("Memento pattern example");
setDefaultCloseOperation(EXIT_ON_CLOSE);
setSize(400,200);
setVisible(true);
}
public void actionPerformed(ActionEvent ae) {
    Command cmd = (Command)ae.getSource();
    cmd.execute();
}
public static void main(String[] args) {
    new MementoDemoDp();
}
}
```

Tips & Tricks

1. Copy content from one file to another

This example explains how to copy contents from one file to another file. Copy file is one of the good use of io package of Java. The logic of program is explained below:

Explanation

This program copies one file to another file. We will be declaring a function called *copyfile* which copies the contents from one specified file to another specified file.

```
copyfile(String srFile, String dtFile)
```

The function *copyfile*(String srFile, String dtFile) takes both file name as parameter. The function creates a new File instance for the file name passed as parameter

```
File f1 = new File(srFile);  
File f2 = new File(dtFile);
```

and creates another InputStream instance for the input object and OutputStream instance for the output object passed as parameter

```
InputStream in = new FileInputStream(f1);  
OutputStream out = new  
FileOutputStream(f2);
```

and then create a byte type buffer for buffering the contents of one file and write to another specified file from the first one specified file.

```
byte[] buf = new byte[1024];  
out.write(buf, 0, len);
```

CopyFile.java:

```
import java.io.*;  
  
public class CopyFile{  
    private static void copyfile(String srFile,  
String dtFile){  
        try{  
            File f1 = new File(srFile);  
            File f2 = new File(dtFile);  
            InputStream in = new  
FileInputStream(f1);  
            OutputStream out = new  
FileOutputStream(f2);
```

```
byte[] buf = new byte[1024];  
int len;  
while ((len = in.read(buf)) > 0){  
    out.write(buf, 0, len);  
}  
in.close();  
out.close();  
System.out.println("File copied.");  
}  
catch(FileNotFoundException ex){  
    System.out.println(ex.getMessage() + "  
in the specified directory.");  
    System.exit(0);  
}  
catch(IOException e){  
    System.out.println(e.getMessage());  
}  
}  
public static void main(String[] args){  
    switch(args.length){  
        case 0: System.out.println("File has not  
mentioned.");  
            System.exit(0);  
        case 1: System.out.println("Destination  
file has not mentioned.");  
            System.exit(0);  
        case 2: copyfile(args[0],args[1]);  
            System.exit(0);  
        default : System.out.println("Multiple  
files are not allow.");  
            System.exit(0);  
    }  
}  
}
```

Output:

```
C:\javajazzup>javac CopyFile.java  
C:\javajazzup>java CopyFile a.java  
Filterfile.txt  
File copied.  
C:\javajazzup>
```

2. Pop-up Menus

A PopupMenu is similar to a Menu as it contains MenuItem objects. The Pop-up Menu can be popped over any component while generating the appropriate mouse event rather than letting it appear at the top of a Frame. Menu class can only be added to a Frame and not to the Applet.

Tips & Tricks

To add it to the Applet you need to use the Swing component set.

In the program code given below, we have used `MouseEvent.isPopupTrigger()` method to trigger the `MouseEvent` that pops up the menu. The example below shows the triggering of a pop-up menu and its activation through a command button.

Code of Program: PopupMenuDemo.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class PopupMenuDemo extends Applet{
    Button b;
    TextField msg;
    PopupAppMenu m;
    public PopupMenuDemo(){
        setSize(200, 200);
        b = new Button("Pop-up Menu");
        add(b, BorderLayout.NORTH);
        msg = new TextField();
        msg.setEditable(false);
        add(msg, BorderLayout.SOUTH);
        m = new PopupAppMenu(this);
        add(m);
        b.addActionListener(new
ActionListener(){
public void actionPerformed(ActionEvent e){
    m.show(b, 20, 20);
    }
});
        addMouseListener(new MouseAdapter(){
            public void mousePressed(MouseEvent e){
                if (e.isPopupTrigger())
                    m.show(e.getComponent(), e.getX(),
e.getY());
            }
            public void mouseReleased(MouseEvent e){
                if (e.isPopupTrigger())
                    m.show(e.getComponent(), e.getX(),
e.getY());
            }
        });
    }
    public static void main(String[] args){
        popupMenuDemo app = new
PopupMenuDemo();
        app.setVisible(true);
    }
}
```

```
    }
}
```

```
class PopupAppMenu extends PopupMenu
implements ActionListener{
    PopupMenuDemo ref;
    public PopupAppMenu(PopupMenuDemo ref){
        super("File");
        this.ref = ref;
        MenuItem mi;
        add(mi = new MenuItem("Copy"));
        mi.addActionListener(this);
        add(mi = new MenuItem("Open"));
        mi.addActionListener(this);
        add(mi = new MenuItem("Cut"));
        mi.addActionListener(this);
        add(mi = new MenuItem("Paste"));
        mi.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e){
        String item = e.getActionCommand();
        ref.msg.setText("Option Selected: " + item);
    }
}
```

PopupMenuDemo.html

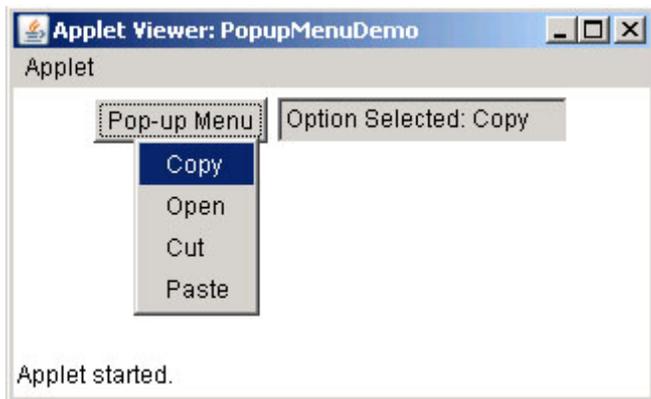
```
<HTML>
<HEAD>
</HEAD>
<BODY>
<APPLET ALIGN="CENTER"
CODE="PopupMenuDemo.class"
WIDTH="800" HEIGHT="500"></APPLET>
</BODY>
</HTML>
```

Run the program:

```
C:\newprgrm>javac PopupMenuDemo.java
C:\newprgrm>appletviewer
PopupMenuDemo.html
```

Tips & Tricks

Output of the program:



3. JSlider Component of Java Swing

A Slider is a Swing tool that lets the user select a value within a bounded range by moving a knob. In this program, events on the JSlider component have also been shown. If you increase or decrease the slider by selecting then the actual position of the slider will be displayed on a label. Some methods and APIs have been used to create a JSlider component and perform various tasks related to the slider. Methods and APIs are as follows:

JSlider :

This class creates the slider for the swing application.

ChangeListener:

This is the interface of which is used to call stateChanged() method which receives the event generated by the slider using addChangeListener() method of the **JSlider** class.

ChangeEvent:

This is the class that handles the event generated by the JSlider component on change the state.

addChangeListener(object):

This is the method of the **JSlider** class which is used to handle event on change the selected state of the JSlider component.

Code of Program: SliderExample.java

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class SliderExample{
    JSlider slider;
    JLabel label;
    public static void main(String[] args){
        SliderExample cs = new SliderExample();
    }

    public SliderExample(){
        JFrame frame = new JFrame("Slider");
        slider = new JSlider();
        slider.setValue(50);
        slider.addChangeListener(new
        MyChangeAction());
        label = new JLabel("JavaJazzUp");
        JPanel panel = new JPanel();
        panel.add(slider);
        panel.add(label);
        frame.add(panel, BorderLayout.CENTER);
        frame.setSize(400, 100);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public class MyChangeAction implements
    ChangeListener{
        public void stateChanged(ChangeEvent
        ce){
            int value = slider.getValue();
            String str = Integer.toString(value);
            label.setText(str);
        }
    }
}
```

Output:



Tips & Tricks

You can slide the knob at any desired position as shown below:



4. Dialog Box In Swing Application

In non-swing application we were using System.in class for input or output some text or numeric values but now in the swing application we can use **JOptionPane** class to show the output or show the message. This way of inputting or outputting works very efficiently in the Swing Applications. The window for showing message for input or output makes your application very innovative.

JOptionPane class is available in the **javax.swing** package. This class provides various types of message dialog box as follows:

1. A simple message dialog box that has only one button i.e. "Ok". This type of message dialog box is used only for showing the appropriate message and user can finish the message dialog box by clicking the "Ok" button.

2. A message dialog box that has two or three buttons. You can set several values for viewing several message dialog box as

follows:

- 1.) "Yes" and "No"
- 2.) "Yes", "No" and "Cancel"
- 3.) "Ok", and "Cancel"

3. An input dialog box that contains two buttons "Ok" and "Cancel".

The **JOptionPane** class has three methods as follows:

showMessageDialog(): First is the **showMessageDialog()** method which is used to display a simple message.

showInputDialog(): Second is the **showInputDialog()** method which is used to display a prompt for inputting. This method returns a String value which is entered by you.

showConfirmDialog(): And the last or third method is the **showConfirmDialog()** which asks the user for confirmation (Yes/No) by displaying message. This method returns a numeric value either 0 or 1. If you click on the "Yes" button then the method returns 1 otherwise 0.

I. Show dialog box.

Message dialog box is used to display informative messages to the user. In the example program, we will use **JOptionPane** class to display the message Dialog box. Our program display "Show Message" button on the window and when user clicks on it program displays Message box with "OK" button and message "JavaJazzUp".

Description:

showMessageDialog():

This method is used to show a message dialog box which contains some text messages. This is being used with two arguments in the program where the first argument is the parent object in which the dialog box opens and another is the message which has to be shown.

Tips & Tricks

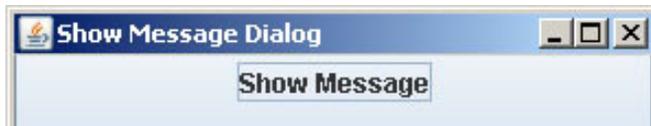
Code of Program: ShowDialogBox.java

```
import javax.swing.*;
import java.awt.event.*;
public class ShowDialogBox{

    JFrame frame;
    public static void main(String[] args){
        ShowDialogBox db = new ShowDialogBox();
    }
    public ShowDialogBox(){
        frame = new JFrame("Show Message Dialog");
        JButton button = new JButton("Show
        Message");
        button.addActionListener(new MyAction());
        frame.add(button);
        frame.setSize(300, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);
    }
    public class MyAction implements
    ActionListener{
        public void actionPerformed(ActionEvent e){
            JOptionPane.showMessageDialog
            (frame,"JavaJazzUp");
        }
    }
}
```

Output:

When you run the program, following window will be displayed:



When you click on "Show Message" button, following Message is displayed:



II. Show message and confirm dialog box

There are three types of message dialog box that you can use in your swing applications. In this section, we will display several types of message boxes. When you run the program, it will display a frame with three buttons.

Description:

1. **showMessageDialog():** Above method shows a simple message dialog box, which holds only one button i.e. "Ok" button. This method takes four arguments in which, first is the parent object name, second is the message as string, third is the title of the message dialog box as string and the last is the type of the message dialog box.

2. **showConfirmDialog():** Above method asks from the user by displaying a message dialog box, which contains more than one button. Depending on the parameter passed it can be "Ok" and "Cancel" or "Yes", "No" and "Cancel". This method returns the integer value.

Code of Program: ShowMessageDialog.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ShowMessageDialog{
    JButton button;
    public static void main(String[] args){
        ShowMessageDialog md = new
```

Tips & Tricks

```
ShowMessageDialog();
}

public ShowMessageDialog(){
    JFrame frame = new JFrame("Message
Dialog Box");
    button = new JButton("Simple message
dialog box");
    button.addActionListener(new MyAction());
    JPanel panel = new JPanel();
    panel.add(button);
    button = new JButton("\Ok/Cancel\
message dialog box");
    button.addActionListener(new MyAction());
    panel.add(button);
    button = new JButton("\Yes/No/Cancel\
dialog box");
    button.addActionListener(new MyAction());
    panel.add(button);
    frame.add(panel);
    frame.setSize(400, 200);
    frame.setVisible(true);
    frame.setDefaultCloseOperation
(JFrame.EXIT_ON_CLOSE);
}

public class MyAction implements
ActionListener{
    public void actionPerformed(ActionEvent
ae){
        String str = ae.getActionCommand();
        if(str.equals("Simple message dialog
box")){
            JOptionPane.showMessageDialog(null,
"Simple message dialog box.", "javajazzup",
1);
        }
        else if(str.equals("\Ok/Cancel\
message dialog box")){
            if(JOptionPane.showConfirmDialog(null,
"\Ok/Cancel\
message dialog box.",
"javajazzup",
JOptionPane.OK_CANCEL_OPTION) == 0)
                JOptionPane.showMessageDialog(null,
"You clicked on \Ok\
button", "javajazzup",
1);
            else
                JOptionPane.showMessageDialog(null,
"You clicked on \Cancel\
button",
"javajazzup", 1);
        }
        else if(str.equals("\Yes/No/Cancel\
message dialog box")){
            JOptionPane.showConfirmDialog(null,
"\Yes/No/Cancel\
message dialog box.",
"javajazzup",
JOptionPane.YES_NO_CANCEL_OPTION) == 0)
                JOptionPane.showMessageDialog(null,
"You clicked on \Ok\
button", "javajazzup",
1);
            else if(str.equals("\Yes/No/Cancel\
message dialog box.") == 1)
                JOptionPane.showMessageDialog(null,
"You clicked on \No\
button", "javajazzup",
1);
            else if(str.equals("\Yes/No/Cancel\
message dialog box.") == 2)
                JOptionPane.showMessageDialog(null,
"You clicked on \Cancel\
button", "javajazzup",
1);
        }
    }
}
```

```
dialog box")){
    JOptionPane.showConfirmDialog(null,
"\Yes/No/Cancel\
message dialog box.");
}
}
}
}
```

Output

If you click on the first button then the simple message box will open which holds only "Ok" button as shown below:

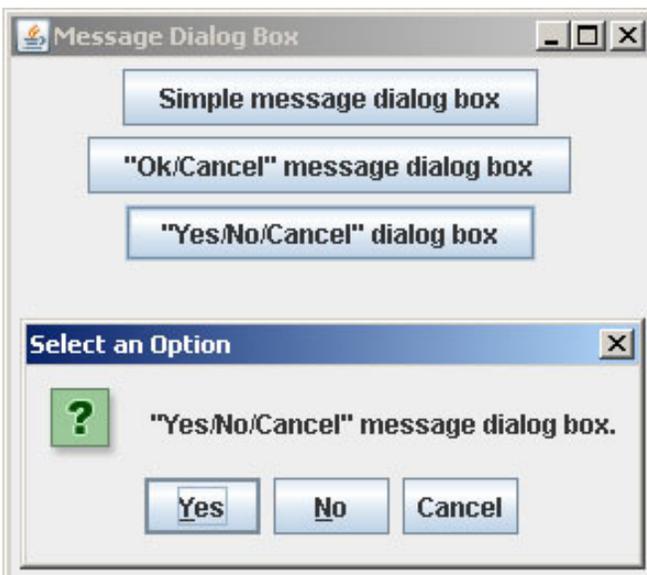


If you click on the second button then the confirm dialog box will open which asks for "Ok" and "Cancel".

Tips & Tricks



If you click on the third button from the main window or frame then a confirm message dialog box will open with three buttons i.e. the "Yes", "No" and "Cancel" like the following image:



III. Show input dialog box

Java Swing provides the way to input any text or numeric value in a normal window (Input Dialog Box). It contains two buttons "Ok" and "Cancel". The program given below shows a button labeled by "Show Input Dialog Box to

enter name" on the frame. Clicking the button opens an input dialog box. If you click on the "Ok" button then a message dialog box appears containing message "Welcome: entered_text" otherwise it displays a message dialog box containing message "You pressed cancel button."

Code of Program: ShowInputDialog.java

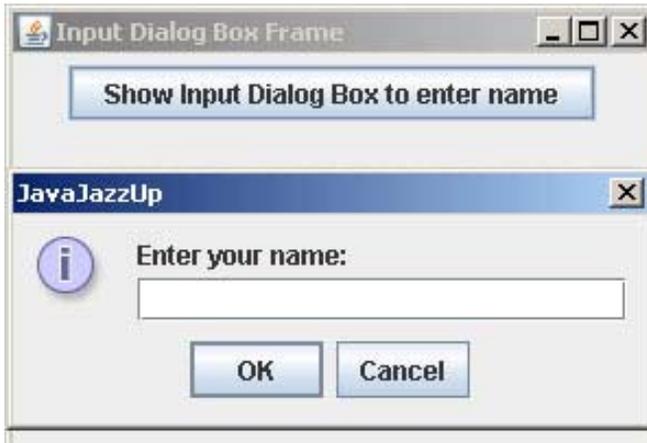
```
import javax.swing.*;
import java.awt.event.*;

public class ShowInputDialog{
    public static void main(String[] args){
        JFrame frame = new JFrame("Input Dialog
        Box Frame");
        JButton button = new JButton("Show
        Input Dialog Box to enter name");
        button.addActionListener(new
        ActionListener(){
            public void actionPerformed(ActionEvent
            ae){
                String str =
                JOptionPane.showInputDialog(null, "Enter
                your name: ",
                "JavaJazzUp", 1);
                if(str != null)
                    JOptionPane.showMessageDialog(null,
                    "Welcome: " + str,
                    "JavaJazzUp", 1);
                else
                    JOptionPane.showMessageDialog(null,
                    "You pressed cancel button.",
                    "JavaJazzUp", 1);
            }
        });
        JPanel panel = new JPanel();
        panel.add(button);
        frame.add(panel);
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(
        JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

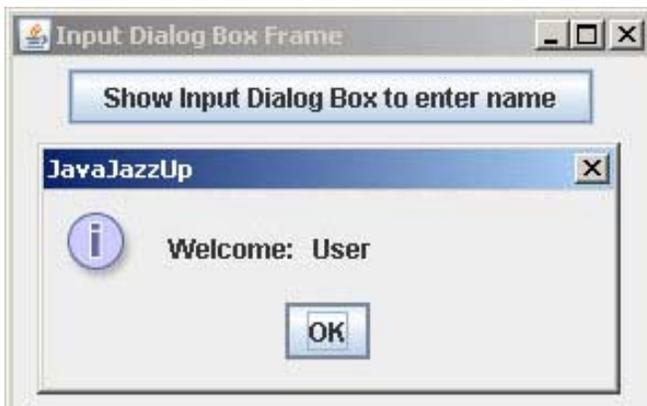
Tips & Tricks

Output:

Clicking "Show Input Dialog Box to enter name" button creates Input Dialog Box as shown below:



If "User" is entered as name in the input box then it welcomes the user with its name as shown below:



India's Cheapest web Service Provider

 **RoseIndia**
Technologies Pvt. Ltd.

Logon to: <http://www.roseindia.net/services/>

OUR SERVICES

- ERP Solutions • Software Solutions • Web Development
- Web Designing • Web Redesigning • Domain Registration
- Web Promotion • Article Writing • Blog Writing
- News Writing • SEO Copywriting • E-Commerce Solutions
- CRM • Outsourcing

Advertise with JavaJazzUp

We are the top most providers of technology stuffs to the java community. Our technology portal network is providing standard tutorials, articles, news and reviews on the Java technologies to the industrial technocrats. Our network is getting around 3 million hits per month and its increasing with a great pace.

For a long time we have endeavored to provide quality information to our readers. Furthermore, we have succeeded in the dissemination of the information on technical and scientific facets of IT community providing an added value and returns to the readers. We have serious folks that depend on our site for real solutions to development problems.

JavaJazzUp Network comprises of :

<http://www.roseindia.net>
<http://www.newstrackindia.com>
<http://www.javajazzup.com>
<http://www.allcooljobs.com>

Advertisement Options:

| Banner | Size | Page Views | Monthly |
|----------------|-----------|------------|-----------|
| Top Banner | 470*80 | 5,00,000 | USD 2,000 |
| Box Banner | 125 * 125 | 5,00,000 | USD 800 |
| Banner | 460x60 | 5,00,000 | USD 1,200 |
| Pay Links | | Un Limited | USD 1,000 |
| Pop Up Banners | | Un Limited | USD 4,000 |

The <http://www.roseindia.net> network is the "real deal" for technical Java professionals. Contact me today to discuss your customized sponsorship program. You may also ask about advertising on other Technology Network.

Deepak Kumar
deepak@roseindia.net

India's Cheapest web Service Provider

Web Packages

Package (in INR)

| Package Name | Price |
|-------------------|-------------------------|
| Starter Package | Rs 5,100 + Taxes Extra |
| Business Package | Rs 9,111 + Taxes Extra |
| Corporate Package | Rs 22,500 + Taxes Extra |
| Smart Package | Rs 45,500 + Taxes Extra |

* Domain Registration free with every package

Packages Specifications

Starter Package

- 5 Web Pages
- 10 Stock Images
- 5 POP 3 Accounts
- 50 MB Hosting Space
- Unlimited Edits

Business Package

- 10 Web Page Design
- Flash Site Animation
- 25 Stock Images
- 10 POP3 Accounts
- 50 MB Hosting Space
- Unlimited Edits

Corporate Package

- 25 Web Page Design
- Flash Site Animation
- Flash Intro Page
- 50 Stock Images
- 25 POP3 Accounts
- 100 MB Hosting Space
- Unlimited Edit

Smart Package

- 200 Web Page Design
- Catalog with 200 items
- Flash Site Animation
- Flash Intro Page
- Unlimited Stock Images
- 50 POP3 Accounts
- 250 MB Hosting Space
- Unlimited Edits

 **RoseIndia**
Technologies Pvt. Ltd.

Logon to: <http://www.roseindia.net/services/>

Valued JavaJazzup Readers Community

We invite you to post Java-technology oriented stuff. It would be our pleasure to give space to your posts in JavaJazzup.

Contribute to Readers Forum

If theres something youre curious about, were confident that your curiosity, combined with the knowledge of other participants, will be enough to generate a useful and exciting Readers Forum. If theres a topic you feel needs to be discussed at JavaJazzup, its up to you to get it discussed.

Convene a discussion on a specific subject

If you have a topic youd like to talk about . Whether its something you think lots of people will be interested in, or a narrow topic only a few people may care about, your article will attract people interested in talking about it at the Readers Forum. If you like, you can prepare a really a good article to explain what youre interested to tell java technocrats about.

Sharing Expertise on Java Technologies

If youre a great expert on a subject in java, the years you spent developing that expertise and want to share it with others. If theres something youre an expert on that you think other technocrats might like to know about, wed love to set you up in the Readers Forum and let people ask you questions.

Show your innovation

We invite people to demonstrate innovative ideas and projects. These can be online or technology-related innovations that would bring you a great appreciations and recognition among the java technocrats around the globe.

Hands-on technology demonstrations

Some people are Internet experts. Some are barely familiar with the web. If you'd like to show others aroud some familiar sites and tools, that would be great. It would be our pleasure to give you a chance to provide your demonstrations on such issues : How to set

up a blog, how to get your images onto Flickr, How to get your videos onto YouTube, demonstrations of P2P software, a tour of MySpace, a tour of Second Life (or let us know if there are other tools or technologies you think people should know about...).

Present a question, problem, or puzzle

Were inviting people from lots of different worlds. We do not expect everybody at Readers Forum to be an expert in some areas. Your expertise is a real resource you may contribute to the Java Jazzup. We want your curiosity to be a resource, too. You can also present a question, problem, or puzzle that revolves around java technologies along with their solution that you think would get really appreciated by the java readers around the globe.

Post resourceful URLs

If you think you know such URL links which can really help the readers to explore their java skills. Even you can post general URLs that you think would be really appreciated by the readers community.

Anything else

If you have another idea for something youd like to do, talk to us. If you want to do something that we havent thought of, have a crazy idea, wed really love to hear about it. Were open to all sorts of suggestions, especially if they promote readers participation.

JOY OF WORK



All Cool Jobs
<http://www.allcooljobs.com/>



BLUR PRINTING |||||

<http://www.newstrackindia.com>