Mar 08 | Volume 1 |Issue 9 | Free

Java Jazz up

A BETTER WAY TO LEARN PROGRAMMING

INTRODUCTION TO XSL

JAXP API USING DOM PARSER

DOJO TUTORIAL

STRUTS 1.1

HIBERNATE QUERY LANGUAGE

VISITOR DESIGN PATTERN

STRUTS 2 NON-FORM TAGS

Tips & Tricks / Advertise With Us

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

Java Jazz up
A BETTER WAY TO LEARN PROGRAMMING

March 2008 Volume I Issue IX

"Optimism with determination lets you hit the goal harder"

Published by

RoseIndia

Editorial

Dear Readers,

We are back here with the Holi (Mar 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 Hibernate 3.0, 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.

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 |** Testing and optimizing Java code without test automation for handling concurrent activities is rather difficult.

12 **Introduction to XSL |** XSL stands for E**X**tensible **S**tylesheet **L**anguage. The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language. Thus it is a language for expressing Stylesheets.

18 **JAXP API using DOM Parser |**In the previous issue of Java Jazz Up, you have read about the JAXP APIs and learned how an XML document is parsed using the serially access mode (SAX) parser.

24 **Struts 1.1 |** This tutorial provides you a better understanding to develop a robust application using Jakarta Struts Framework.

29 **Struts 2 Non-form Tags (UItags) |**Apache Struts is an open-source framework used to develop Java web applications. In this section, struts 2 non-form tags (UItags) will be discussed.

36 **Design Pattern |** These types of design patterns are used as templates. These design patterns are used in such conditions when we need a parent class having one or more methods to be implemented by their child classes.

38 **Visitor Design Pattern|** The design pattern provides additional functionality to a class. The Visitor pattern allows us to create an external class to act on data in other classes.

40 **Dojo Tutorial| Dojo:** Dojo is an Open Source JavaScript toolkit libraries that provides a simple API(Application Programming Interface) for building the serious applications in less time.

45 **Hibernate Query Language|** In the previous issue of Javajazzup you learned about Hibernate Query Language and its different kind of clauses. Lets quickly focus on the overview of HQL.

52 **Using the Desktop class to launch a URL with default browser in Java|** This article describes the new **Desktop** API, which allows Java applications to interact with the default applications associated with specific file types on the host platform.

55 **Advertise with Us |** 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.

57 **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

Concurrency Testing in Java Applications

Testing and optimizing Java code without test automation for handling concurrent activities is rather difficult. Even with test automation, being able to correlate the test activity from the client side to observations of thread, memory, object and database connection use on the server side is difficult at best. This tool describes methods to test concurrency in Java applications and also displays the new technique for correlating about the task that a Java application server is doing on the server side however a load test automation tool drives a test on the client side.

Most of the IT managers consent about the concurrency testing that it is the right way to determine many performance bottlenecks, resource contention issues, and service interruptions. However, only few of the developers use concurrency testing because the available test tools are not satisfactory.

New Features in EJB 3.1

New features are added to the EJB 3.1. Experts are trying to make changes to the EJB 3.1 for the next version of the Java EE specification. The idea behind these changes is to provide the head's up on the changes as well as gather your feedback early so the expert group has the best chance of getting it right. EJB 3.0 is made simple to Java EE 5 by moving away from a heavyweight-programming model. EJB 3.1 targets to build the successes movement down the path of simplicity along with a handful of much-needed features.

The features added to EJB 3.1 makes the interfaces optional for EJBs and Singleton Beans, but none of the features has been finalized yet; all of this is really just a peek into the inner workings of the JCP so that you have a chance provide early feedback.

1. EJB Interfaces are Optional

In EJB 3.1, now you do not need to define any interfaces for Session Beans, just like JPA Entities and Message Driven Beans. All you have to do is annotate a POJO with the @Stateless or @Stateful to get a fully functional EJB.

2. The Singleton Beans

A new feature of Singleton Beans is added in EJB 3.1 that is used to store application-wide shared data. The JEE container maintains a single shared instance of an EJB 3.1 Singleton that can cache state across the application tier. Like all other EJBs, Singletons are simply annotated POJOs.

3. Support for direct use of EJBs in the servlet container, including simplified packaging options. Like the web.xml file that resides in the WEB-INF directory, you would be able to place an EJB jar into the WEB-INF/lib directory.

4. Support for stateful web services via Stateful Session Bean web service endpoints.

Load Balancing Tomcat with Apache

Tomcat is the most popular application server that is used for hosting web applications. Apache is also popular web server that enables services such as https encryption and decryption, URL rewriting etc. Apache also serves as a load balancer for balancing the load between several Tomcat application servers.

Profiles in the Java EE 6 Platform

Profiles are an attempt to modularize the different parts in Java EE so that the technologies can be combined in a product. This will help in solving the issue of compatibility requirements in order to provide the natural approach by spanning the multiple technologies. E.g: Java EE 5 specification contains requirements to which the servlet containers must honor with respect to the availability of JTA. The idea behind Java EE 6 is to rewrite such type of requirements, which can be applied to any profiles and implementations, concerning the relevant combination of technologies. E.g: any product corresponding to any Java EE profile including servlets as well as JTA will have to be sincere to those requirements. The logic includes two components: the first one is that, we think that the Java EE requirements add significant value to standalone technologies, as testified e.g. by the large number of servlet containers implementing JTA so that it is compatible with what Java EE mandates; simultaneously, calling out the requirements that helps to ensure that applications that target profiles will work on the full platform.

This whole phenomena makes profiles more than just collections of independently tested technologies, as those technologies will be tied together in interesting ways, deliver more functionality than they would on their own.

Apache Geronimo 2.1 Released -- Java EE 5 server

Recently, Apache Geronimo team announced the release of Apache Geronimo 2.1.
Apache Geronimo 2.1 is developed on the Java EE 5 certified 2.0 release of Geronimo. Geronimo 2.1 provides the following new features:

Custom Server Assemblies:
Geronimo 2.1 greatly simplifies the build-time customization. It allows the users to follow a function-centric approach, choosing the desired set of server plugins (e.g. Web Container+JMS+Deploy capabilities).

Flexible Administration Console:
Now the Geronimo Administration Console matches the dynamic capabilities of the server runtime.

Gshell:
It is a command-line processing environment required to execute Geronimo administrative commands.

WADI Clustering:
Now WADI enables to cluster Web Applications for both Tomcat and Jetty-based configurations of Geronimo.

Comet: Reverse Ajax for streaming data from the server

Comet is a technology that pushes the events from the server side to a browser client. It avoids the issues related to having a browser poll a server to check for new events. Comet looks at the nature of real-time events, such as those occurring in the stock market or in sporting matches. Rich-client technologies such as Flash or Java this process seems original particularly since these last technologies have combined mechanisms such as remoting and steaming that deals with asynchronous requests from both server to client and vice versa. But Comet is all about concerning the last process having the same vanilla technologies like Ajax namely JavaScript and

HTML.
However concerning with the actual term, Comet is different to Ajax's meaning outside IT circles as a household cleaner. But sudden name or not, Comet serves the purpose of an umbrella name for delivering data onto browsers as it becomes available on the serverside, a technique that will surely be of the same impact and go hand in hand with what we know today as Ajax.

MyEclipse Blue Edition: Low-Cost Tool Alternative for WebSphere

Genuitec announced the new release of a product, MyEclipse Blue Edition that targets IBM's Rational Application Developer (RAD) and WebSphere development.

With ending support to WebSphere Application Developer (WSAD), all WebSphere users necessitates to upgrade their toolsuites to IBM RAD so that they can support the latest features in the WebSphere 6.1 server that is a transition having high cost in terms of time and hard currency. Since many of these IBM shops are necessarily requiring changing their tool environments, Genuitec has offered a choice to the consumers.

MyEclipse 6.1 Blue Edition is the new edition offering a complimentary toolsuite for a tiny fraction of the cost to the RAD users. This new Blue Edition has a cost only $149 per year, and provides a full support. You donot need to worry about nullifying your existing support contracts. You're free to use as MyEclipse Blue Edition uses IBM's own Web Services. MyEclipse 6.1 Blue Edition is in Milestone stage 1, and provided as a tool to download and use upto April 1st.

App Performance Management Scenario: Changing Java Developer's Role

Developers are spending most of their time in maintaining existing applications, instead of actually developing new features. This changes the scenario of development that tends to be a little more workmanlike than we'd like to admit sometimes.

This is not the new task as most of the developers rarely admit the new development, even in the new projects; new features are created as if they were being built in maintenance mode instead of being made out of whole cloth.

A lot of tools exist to impose the specific problems therefore most of the application management lifecycle are spending their time in finding specific problems.

Two processes avoid most of the maintenance work while the SSQ article mentions only one of them.

The first maintenance obstacle is testing – The whole application can be tested, as the application failures are the larger aspect of application maintenance.

The other maintenance obstacle is performance – performance is managed in a better way by assuming it as an attribute of the code throughout development, rather than deploying and testing the application and finding out after "completion" that it doesn't meet its performance requirements.

Agile development helps by forcing for more tests throughout the entire lifecycle; using Java application servers helps in isolating the performance issues as they provide a central node for monitoring, While there are many of the performance tools available for Java leverage the application server and its APIs in just this fashion, by monitoring the boundaries between APIs.

FindBugs™ Released new version: Version 1.3.2

FindBugs™ recently released a new version: Version 1.3.2.

This is available for download on sourceforge.net. and can be installed.

It is a better static analyzer identify a variety of bugs and potential bugs in Java code.
It can analyze class files and/or source files.

It provides a bug tree. User can select an item in the bug tree and may get an explanation in the text panel.

FindBugs quickly provides an overview of items required by the programmer in the java code.

New Networking features with PPF grid toolkit: Releases in v1.1

Recently JPPF i.e. Java Parallel Processing

Framework was released with version 1.1. New features of JPPF:

Several bugs are fixed

- Provides a new networking tool i.e. the TCP port multiplexer that allows it even to work in the fire walled environments
- Addition of new node monitoring feature
- PPF grid takes no time to be up and run
- provides highly scalable, distributed framework for the execution of Java tasks.
- Better graphical and programmatic tools
- reliability through redundancy
- failover recovery capabilities

Apache Tuscany SCA Java 1.1 -incubating released

Apache Tuscany team has announced the release of the Java SCA project of version 1.1. It provides a runtime environment based on SCA (Service Component Architecture), which is a set of specifications to simplify SOA application development, standardized by OASIS.

This release has added features like JMS binding, improved policy support and an implementation extension for representing client side JavaScript applications as SCA components.

New Run Time Editor 3 (RTE3) for Java Released

The new RTE3 is the control system interface design environment that has been designed for network control in java. RTE3 is fully written in Java so it is very expandable. It is available for Windows and Linux environment.

RTE3 provides some new capabilities like multi-threaded, multi-processor aware, easy socket communication, MP3 and MPEG capabilities, access dlls, shell scripting, unlimited run time editing/ scripting, multiple windows, multiple pages, grouping, group editing, built-in database access, unlimited undo/redo etc.

Theory and Practice of Ropes for Java String Manipulations

Java language's default String and StringBuilder classes poorly serves to large quantities of data manipulated by the Systems. A rope data structure may be a better alternative. A rope implementation for the Java platform; provides pointers for effective use of the library and defines performance issues. Iteration over a flat Rope (a Rope having depth of 1) is much faster than pulling data character by character from a String is the most considerable thing for an enterprise Java developer. If you do not need to do this, then Ropes are completely meaningless to you.

Carlos Perez: Top Five Java Technologies to Learn in 2008

Carlos Perez has posted a list of the top five Java-based technologies to learn in 2008.

They are:

- OSGi, a specification for dynamic modules for Java
- The Java Content Repository spec, appeared first time in the JCP in February 2002
- Google Web Toolkit, first version released in May, 2006
- Groovy, first version released in May, 2004
- Cloud computing, a concept designed around the use of virtual servers, or distributed computing without using EJB

However all the above technologies are holder technologies perhaps are "coming of age" and became matured to the point of recommendation. But definitely all these technologies are still useful as OSGi works as module system behind Eclipse, while Groovy is continuously accepting by the developer with its formal specification and continually improved releases, GWT is already mature and stable, and cloud computing is becoming more and more famous in this growing marketplace.

On the other hand, JCR and cloud computing are the least accepted of these technologies, and vendors are trying to address that, with competitions to spur awareness or active community involvement.

Apache Jackrabbit 1.4 released

Apache Jackrabbit 1.4 is the latest and greatest version of Content Repository for Java Technology API (or JCR in short) conforming that Apache Jackrabbit 1.4 provides the full open source implementation of JCR.

JCR is a standard API to access the content repositories in a uniform manner and is specified in JSR 170. A content repository is a hierarchical representation storage media similar to an advanced file system supporting different levels of content structure and granularity. The API provides various functionalities such as browsing, modifying, and searching the content in full text search in a content repository. Standard allows advanced features like versioning, observation, locking, and XA transactions as optional.

Apache Jackrabbit provides one of the best JCR implementations and was also used as the reference implementation of JSR 170. Apache Jackrabbit 1.4 is a stable and feature-rich content repository aimed having a wide range of content applications including Magnolia, OpenKM, Hippo, and Mindquarry. Jackrabbit provides implementation for all the mandatory and optional JCR features as well as a number of extensions and related JCR tools.

Apache Jackrabbit is the biggest ever release having 220 new features, improvements and bug fixes on the basis of feedback and contributions from the user community.

Sun Microsystems has agreed to buy MySQL AB for $1B

Sun Microsystems has acceded to buy MySQL AB for $1B, by providing additional leverage in the open source community and also allowing access to MySQL to its larger corporations. The grand question that arises is that: what does this mean in the long term? Sun already offers a small-scale database as compared to compared to 'large offerings' like Oracle9 and IMS. Is Sun considerably looking for an additional revenue stream from MySQL AB's customers, or shifting away from JavaDB/ Derby?

JVM Lies: The OutOfMemory Myth

The concept "JVM Lies: The OutOfMemory Myth," tells about the happening when a JVM throws an OutOfMemoryError – the developers who have encountered it have noticed, it seems like it's out of memory, but it always doesn't look like it, and throws more RAM at the JVM may help, but that's the wrong solution.

Shades of HotJava: LoboBrowser, a web browser in Java

JavaLobby have introduced a LoboBrowser, a fully Java web browser. Although it is not HotJava, yet it runs JavaScript if memory is available while HotJava did not. It is big deal to develop a workable and installable fully Java based browser even if it can't render TSS very well. It also includes a rendering engine, Corba that can be used to watch DOM of a page even after JavaScript has been run over it. Integration into IDEs, can be seen to anyone. E.g providing browser support without allowing explicit browser tie-ins.

Java News and Releases

ServletExec 6.0 Released

New Atlanta has released ServletExec 6.0 for download and purchase:

http://www.newatlanta.com/products/servletexec

Key new features contained in ServletExec 6.0 are:

- Java Servlet API 2.5
- JavaServer Pages (JSP) 2.1
- JSP Standard Tag Library (JSTL) 1.2
- JavaServer Faces (JSF) 1.2
- JavaMail 1.4
- Java Web Services support through JAX-WS 2.0
- Updated Web Server support (IIS 7, Apache 2.2.x, SJSWS 7.0u1)
- Updated OS support (Windows 2008 Vista, Solaris 10, AIX 5.3, HP-UX 11v2)
- AMD/Intel 64-bit support (x64)

Additional improvements are:

- Improved Performance
- IPv6 Protocol support
- Improved Administrative Interface

Important notes:

- JDK/JRE 1.5 or 1.6 is required for ServletExec 6.0
- New license for ServletExec 6.0 will be provided to the existing customers with current subscriptions free of charge
- Customers who have already purchased ServletExec 5.x on or after November 1st, 2007 will have the facility to upgrade to ServletExec 6.0 free of charge

ServletExec is one of the original implementations of the Java Servlet API and around 14000 customers in 85 countries has been purchased to this original implementation.

Java Remoting: Protocol Benchmarks

Every client/server application may have different remoting requirements, but the main criteria include performance. At least, you would to know that how much performance you are sacrificing in order to fulfilling other requirements.

Java Remoting: Protocol Benchmarks examines Java's RMI/JRMP, Spring's HttpInvoker, Oracle's ORMI (with and without HTTP tunneling enabled), and three flavors of Apache XML-RPC and Caucho's Hessian, Hessian 2 and Burlap. Two graphs are added named as a small list having 250 items or less and a large list having from 500 to 5000 items. The two graphs are consistent (a good sign for the libraries involved) - and (surprise!) the binary protocols did far better on average than the XML-based protocols.

Article: Integrating Java and Erlang

Enterprise software development world introduces a new face: Erlang. Erlang is nothing but a functional programming language having native constructs for concurrency and reliability. Erlang is a programming language and Jinterface is an open source component of Ericsson's Open Telecom Platform having capability to integrate Java with Erlang.

Erlang provides a trivial solution for Building scalable, distributed and reliable systems. It may be a little bit hard to swallow for many enterprise developers. Erlang is a dynamically typed functional language; on the other hand Java is a statically typed object-oriented language. Erlang works as a agitator in case of traditional enterprise development as it is an standard similar to concurrency, uptimes of five nines or more, and "hot deployment" of code.

Java Persistence API (JPA) implementation for Amazon SimpleDB

JPA provides implementation for SimpleDB. A SimpleDB is webservice for running queries on structured data in real time. The current version supports the following features:

- @ManyToOne - object references.
- @OneToMany - collections.
- @MappedSuperClass - Useful timestamping class and for a common ID.
- @Inheritance
- @Lob - stores lobs in S3 so make it as big as you want (up to 5GB).
- @Id - Assigns UUID automatically
- Lazy loading on ManyToOne, Lobs and OneToMany so only hits SDB on an as needed basis.
- Rudimentary caching to reduce hits on SDB
- JPA Queries
- Perst All-Java Embedded Database Adds KD-Tree Indexes

McObject provides support for the KD-Tree. A KD-Tree is a database index that is used in spatial and pattern-matching applications, to Perst. It is an open source, object-oriented all-Java embedded database system. Developers who have been working with Perst, the KD-Tree helps them in expanding coding efficiency and helps in making Java data objects easier to use in certain types of application.

The new k-dimensional tree or in short KD-Tree index enables us to add a structure in Perst for storing and manipulating point objects in a k-dimensional space by partitioning that space. These are practically used in computer graphics, geographical information systems and biometric applications like fingerprint matching as well as their efficiency in handling multi-dimensional data, KD-trees helps the developers in "normal" applications where query predicates include different combinations of object fields.

McObject's Perst embedded database and eXtremeDB embedded database provides a collection various index types, including:

- B-trees used in common sorting, searching, insertions, and deletions
- R-trees used for geospatial indexing (mainly in GPS/navigation systems)
- T-trees used in all-in-memory data storage and accessHash tables for quickly locating a single unique index entry
- Patricia trie index, that is used for speedly searching in networking and telephony applications
- Custom indexes used in b-trees allowing to the application to define the collating sequence of entries; it is also useful in implementing soundex algorithm searches, for example
- Bit or bitmap used for optimizing the indexes for columns in which values repeat frequently
- TimeSeries class used in dealing efficiently with small fixed-size objects
- Specialized versions of collections used in thick indices (indexes having multiple duplicate values), and bit indexes
- KD-Tree is developed by using the latest versions of Perst for Java and .NET. It is available for free download at McObject's Web site.

Introduction to XSL

Introduction to XSL

XSL stands for E**X**tensible **S**tylesheet **L**anguage. The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language. Thus it is a language for expressing Stylesheets.

A stylesheet specifies the presentation of XML information using two basic categories of techniques:

- An optional transformation of the input document into another structure.
- A description of how to present the transformed information.

The components of the XSL language
The full XSL language logically consists of three component languages, which are described in three W3C (World Wide Web Consortium) Recommendations:

- **XPath:** XML Path Language is an expression language used by XSLT to access or refer specific parts of an XML document

- **XSLT:** XSL Transformations is a language for describing how to transform one XML document (represented as a tree) into another.

- **XSL-FO:** Extensible Stylesheet Language Formatting Objects is a language for formatting XML documents and Formatting Properties.

Understanding XSL Stylesheet Structure
(a) XSLT namespace

The XSL stylesheet starts with the root element <xsl:stylesheet> or <xsl:transform> that declares the document to be an XSL style sheet.

The correct way to declare an XSL style sheet according to the W3C XSLT

Recommendation is:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
```

or:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
```

Since an XSL style sheet is an XML document itself, it always begins with the XML declaration: **<?xml version="1.0" ?>**

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.
The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute version="1.0". This specification uses a prefix of xsl: for referring to elements in the XSLT namespace. However, XSLT stylesheets are free to use any prefix.

Now set it up to produce HTML-compatible output:

```
<xsl:stylesheet
 ...
 >
 <xsl:output method="html"/>
 ...
</xsl:stylesheet>
```

(b) Stylesheet Element

The <xsl:template> Element
An XSL style sheet consists of a set of rules that are called templates. Each template "matches" some set of elements in the source tree and then describes the contribution that the matched element makes to the result tree. Most templates have the following form:

```
<xsl:template match="/">
   <html><body>
    <xsl:apply-templates/>
   </body></html>
 </xsl:template>
```

Before processing can begin, the part of the XML document with the information to be copied to the output must be selected with an **XPath** expression. The selected section of the document is called a **node** and is normally selected with the **match** operator.

In the above statements, the **<xsl:template>** element defines a template. The **match="/"** attribute associates the template with the **root node** of the XML source document. Another approach is to match the **document element** (the element that includes the entire document).

The <xsl:apply-templates> Element

The <xsl:apply-templates> element applies a template to the current element or to the current element's child nodes. If we add a select attribute to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed.

The <xsl:value-of> Element

The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation. For example, the given expression will select the value of Emp_Id attribute of the specified element and write to the output:

```
<xsl:value-of select="Emp_Id"/>
          or
<xsl:value-of select="Employee-Detail/
Employee/Emp_Id"/>
```

Note: The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

The <xsl:for-each select="elementName "> Element

The 'for-each' expression is a loop that processes the same instructions for these elements. The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set. For example, the given expression finds all 'Employee' elements in the 'Employee-Detail' element context using the XPath expression 'Employee-Detail/ Employee'.

If the selected node contains all elements in the root, all of the 'Employee-Detail' elements will be selected.

```
<xsl:for-each select="Employee">
<xsl:value-of select="Emp_Id"/>
<xsl:value-of select="Emp_Name"/>
</xsl:for-each>
```

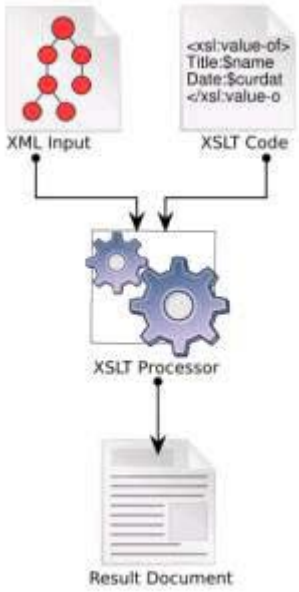
Introduction to XSLT

Extensible Stylesheet Language Transformations (XSLT) is an XML-based language that transforms an XML documents and generates output into a different format such as HTML, XML or another type of document that is recognized by a browser like WML, and XHTML.

XSLT is an extension of XSL, which is a stylesheet definition language for XML.
With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, and make decisions about which elements to hide and display.

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

The following figure shows the working process of XSLT:



XSLT Processors

The job of an XSLT processor is to apply an XSLT stylesheet to an XML source document and produce a result document, (for example HTML document). There are several XSLT processors, but a few good one (Open sources), such as MSXML4, Saxon, and Xalan, XT, Oracle. Most of them can be downloaded free from Web sites.

Apache's Xalan XSLT engine

Xalan is the Apache XML Project's XSLT engine. This processor is available at http://xml.apache.org/xalan/. We will concentrate on using this engine for transformation our XML document that we have developed and want to transform it into output document in the HTML format.

Once the Xalan.zip or .gzip file is downloaded, unpack it and add these files to your CLASSPATH. These files include the .jar file for the Xerces parser, and the .jar file for the Xalan stylesheet engine itself. The .jar files are named **xercesImpl.jar**, and **xalan.jar**.

Working with XSLT APIs

XSLT consist of three components that transform an XML document into the required format. These components are:

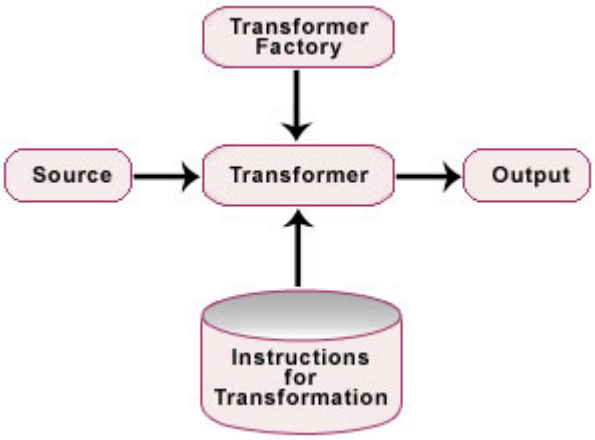
- An instance of the TransformerFactory
- An instance of the Transformer
- The predefined transformation instruction

TransformerFactory is an abstract class used to create an instance of the Transformer class that is responsible for transforming a source object to a result object.

The process of XML transformation starts when you create an instance of the **TransformerFactory** class. An instance of the **Transformer** class is then created using the instance of the TransformerFactory class. This instance of the Transformer class uses the XML document as a source object and optionally uses the predefined instructions required for transformation to generate the formatted output as a result object. You can create the source XML document using SAX, DOM, or an input stream. The result object of the transformation process is in the form of a SAX event handler, DOM, or an output stream.

During transformation process, the original document is not changed; rather, a new document is created based on the content of an existing one. The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text.

The following figure shows the working process of XSLT APIs:



The XSLT Packages

The XSLT APIs is defined in the following packages:

Package	Description
javax.xml.transform	Defines the TransformerFactory and Transformer classes. These classes are used to get an object for doing transformations. After creating a transformer object, its transform() method is invoked. This method provides an input (source) and output (result).
javax.xml.transform.dom	Defines classes used to create input and output objects from a DOM.
javax.xml.transform.sax	Defines classes used to create input from a SAX parser and output objects from a SAX event handler.
javax.xml.transform.stream	Defines classes used to create input and output objects from an I/O stream.

In this tutorial, we will convert a simple XML file into HTML using XSLT APIs.
To develop this program, do the following steps:

1. Create an XML file

The code for the **emp.xml** file is given below:

```
<?xml version = "1.0" ?>

<Employee-Detail>

<Employee>
<Emp_Id> E-001 </Emp_Id>
<Emp_Name> Nisha </Emp_Name>
<Emp_E-mail> Nisha1@yahoo.com </
Emp_E-mail>
</Employee>

<Employee>
<Emp_Id> E-002 </Emp_Id>
<Emp_Name> Amit</Emp_Name>
<Emp_E-mail> Amit2@yahoo.com </Emp_E-
mail>
</Employee>

<Employee>
<Emp_Id> E-003 </Emp_Id>
<Emp_Name> Deepak </Emp_Name>
<Emp_E-mail> Deepak3@yahoo.com </
Emp_E-mail>
</Employee>

</Employee-Detail>
```

2. Create an XSL Stylesheet

Lets see the source code of XSL stylesheet (**emp.xsl**) that provides templates to transform the XML document:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://
www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="html" indent="yes"/>
```

```
<xsl:template match="/">
<html>
<title>XSLT Style Sheet</title>
<body>
<h1><p align="center">Employee Details</
p></h1>
     <xsl:apply-templates/>
     </body>
  </html>
</xsl:template>

<xsl:template match="Employee-Detail">

<table border="2" width="50%"
align="center">
<tr bgcolor="LIGHTBLUE">
<td><b>Emp_Id</b></td>
<td><b>Emp_Name</b></td>
<td><b>Emp_E-mail</b></td>
</tr>
  <xsl:for-each select="Employee">
  <tr>
     <td><i><xsl:value-of select="Emp_Id"/
></i></td>
      <td><xsl:value-of select="Emp_Name"/
></td>
  <td><xsl:value-of select="Emp_E-mail"/
></td>
  </tr>
  </xsl:for-each>
  </table>

</xsl:template>

</xsl:stylesheet>
```

3. Create a Java program using XSLT APIs

Now we will develop a class in Java that takes both XML and XSL file as an input and transforms them to generate a formatted HTML file.

Here is the source code of the **SimpleXMLTransform.java:**

```
import javax.xml.transform.ErrorListener;
import javax.xml.transform.Transformer;
import javax.xml.transform.Transformer
ConfigurationException;
import
javax.xml.transform.TransformerException;
import
javax.xml.transform.TransformerFactory;
import
javax.xml.transform.stream.StreamResult;
import
javax.xml.transform.stream.StreamSource;

public class SimpleXMLTransform {
   static public void main(String[] arg) {
       if(arg.length != 3) {
          System.err.println("Usage:
SimpleXMLTransform " +
             "<input.xml> <input.xsl>
<output>");
          System.exit(1);
       }
       String inXML = arg[0];
       String inXSL = arg[1];
       String outTXT = arg[2];

       SimpleXMLTransform st = new
SimpleXMLTransform();
       try {
          st.transform(inXML,inXSL,outTXT);
       }
catch(TransformerConfigurationException e) {
          System.err.println("Invalid factory
configuration");
          System.err.println(e);
       } catch(TransformerException e) {
          System.err.println("Error during
transformation");
          System.err.println(e);
       }
   }
   public void transform(String inXML,String
inXSL,String  outTXT)
             throws
TransformerConfigurationException,
                TransformerException {

       TransformerFactory factory =
TransformerFactory.newInstance();

       StreamSource xslStream = new
StreamSource(inXSL);
       Transformer transformer =
factory.newTransformer(xslStream);
       transformer.setErrorListener(new
MyErrorListener());
```

```
        StreamSource in = new
StreamSource(inXML);
        StreamResult out = new
StreamResult(outTXT);
        transformer.transform(in,out);
     System.out.println("The generated HTML
file is:" + outTXT);
    }
}
class MyErrorListener implements
ErrorListener {
    public void warning(TransformerException
e)
                throws TransformerException {
        show("Warning",e);
        throw(e);
    }
    public void error(TransformerException e)
                throws TransformerException {
        show("Error",e);
        throw(e);
    }
    public void fatalError(TransformerException
e)
                throws TransformerException {
        show("Fatal Error",e);
        throw(e);
    }
    private void show(String
type,TransformerException e) {
        System.out.println(type + ": " +
e.getMessage());
        if(e.getLocationAsString() != null)
        System.out.println(e.getLocationAsString());
    }
}
```

This program uses three arguments to take inputs from the command line: **arg[0]** is for XML file, **arg[1]** is for XSL file, and **arg[2]** is for taking the name of the html file that will be generated after the transformation.

As in the earlier section, we have described the working process of XSLT APIs. First, this program creates an instance of the **TransformerFactory** class. The new instance of the **Transformer** class is created using an "**xslStream"** instance of the **StreamSource** class. This instance of the Transformer class required for transformation to generate the formatted output as a result object. Its method **transform(in,out)** takes two arguments: the **XML** document as a source object and the result document as an output object in the form of **HTML**.

4. Compile and Run the Program

```
C:\nisha\xslt>javac SimpleXMLTransform.java

C:\nisha\xslt>java SimpleXMLTransform
emp.xml emp.xsl emp.html
The generated HTML file is:emp.html
```

The format of the generated output file **"emp.html"** will look like this:

Employee Details

Emp_Id	Emp_Name	Emp_E-mail
E-001	Nisha	Nisha1@yahoo.com
E-002	Amit	Amit2@yahoo.com
E-003	Deepak	Deepak3@yahoo.com

JAXP API using DOM Parser

In the previous issue of Java Jazz Up, you have read about the JAXP APIs and learned how an XML document is parsed using the serially access mode (SAX) parser. Now you will learn how the DOM parser works with the same xml document. Lets quickly focus on the overview of XML parser.

Introduction to XML Parser:

In computing terms, a parser is a program that takes input in the form of sequential instructions, tags, or some other defined sequence of tokens, and breaks them up into easily manageable parts.

XML parser is used to read, update, create and manipulate an XML document. Whenever the XML document executes, the parser recognizes and responds to each XML structure taking some specified action based on the structure type.

XML parsers can be validating or nonvalidating. Validating parser checks the contents of a document against a set of specific rules i.e. in what order they must appear. These rules appear in an XML document either as an optional XML structure called a document type definition, or DTD, or as an XML Schema.

Nonvalidating parsers are smaller and faster, but they do not check documents against the DTD. They only check whether the XML document is structurally well formed or not.
Parsing XML Documents

To manipulate an XML document, XML parser is needed. The parser loads the document into the computer's memory. Once the document is loaded, its data can be manipulated using the appropriate parser.

In this section, we will discuss about DOM parsers of JAXP APIs and for accessing XML documents in random access mode. The specifications to ensure the validity of XML documents are DTDs and the Schemas.

DOM (Document Object Model)

The XML Document Object Model (XML DOM) defines a standard way to access and manipulate XML documents using any programming language (and a parser for that language).

The DOM presents an XML document as a tree-structure (a node tree), with the elements, attributes, and text defined as nodes. DOM provides access to the information stored in your XML document as a hierarchical object model.

The DOM converts an XML document into a collection of objects in an object model in a tree structure (which can be manipulated in any way). The textual information in XML document gets turned into a bunch of tree nodes and a user can easily traverse through any part of the object tree, any time. This makes easier to modify the data, to remove it, or even to insert a new one. This mechanism is also known as the random access protocol.

DOM is very useful when the document is small. DOM reads the entire XML structure and holds the object tree in memory, so it is much more CPU and memory intensive. The DOM is most suited for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user.

The Document Object Model implementation is defined in the following packages:

JAXP APIs	Description
org.w3c.dom	Defines the Document class (a DOM) along with the classes for all of the components of a DOM.
javax.xml.parsers	The JAXP APIs provide a common interface for different vendors' to use SAX and DOM parsers.

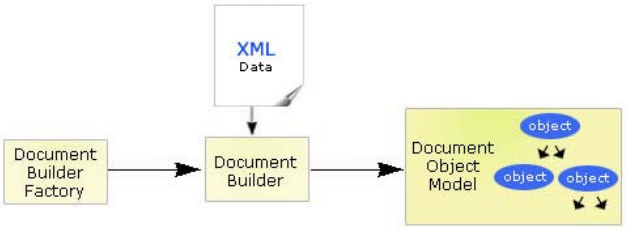
The DOM API is defined in org.w3c.dom package of JAXP-APIs. The DOM API is easier to use. It provides a tree structure of objects. The DOM API is used to manipulate the hierarchy of application objects it encapsulates.

JAXP API using DOM Parser

Main classes of **javax.xml.parsers** package for the DOM:

Classes	Description
DocumentBuilder	Defines the API to obtain DOM Document instances from an XML document.
DocumentBuilderFactory	Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents

The diagram here shows the **JAXP API**s to process xml document using the **DOM parser**:



Understanding DOM Parser

At the very first, **javax.xml.parsers.DocumentBuilderFactory** class creates the instance of **DocumentBuilder**. Through which it produces a Document (a DOM) that conforms to the DOM specification. The System property determines the builder at the run time using javax.xml.parsers.DocumentBuilderFactory (it selects the factory implementations to produce the builder). The platform's default value i.e. system property can be overridden from the command line.

Another method of DocumentBuilder as newDocument() can also be used implementing org.w3c.dom.Document interface that creates an empty Document.

Alternatively, one of the builder's parse methods can be used to create a Document from existing XML data. As a result, a DOM tree like that shown in the diagram.

Creating Blank DOM Document
The following code creates a blank document:

```
//Create instance of DocumentBuilderFactory
    DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
    //Get the DocumentBuilder
    DocumentBuilder parser =
factory.newDocumentBuilder();
   //Create blank DOM Document
        Document doc =
parser.newDocument();
```

The class **DocumentBuilderFactory** is responsible for creating new DOM parsers. The instance of the class **DocumentBuilder** is used to create a blank document. The **newDocument()** method of the class returns a blank DOM document.

Creating DOM Child Elements

Once new document has been created, you can add the element to it. First element of the document is called the **root** element; another elements added to the document are called **child** elements of the root.
The following steps generate a DOM document having root and child elements.

(1) Creating the root element

As you have seen above that how to create a blank DOM document using **DocumentBuilder** object. The following code creates a blank document.

```
//Create blank DOM Document
Document doc = docBuilder.newDocument();
```

The **createElement** method is used to create the root element and **appendChild** method is

used to append the element to the DOM document.

```
//create the root element
Element root = doc.createElement("root");
//all it to the xml tree
doc.appendChild(root);
```

(2) Adding Comment Element to DOM Tree

The **doc.createComment** method is used to reate Comment object.

```
//create a comment
Comment comment =
doc.createComment("This is comment");
//add in the root element
root.appendChild(comment);
```

(3) Adding Child Element to DOM Tree

The **doc.createElement** method is used to create Child element.

```
//create child element
Element childElement =
doc.createElement("Child");
//Add the atribute to the child
childElement.setAttribute("attribute1","The
value of Attribute 1");
root.appendChild(childElement);
```

(4) Printing the DOM Tree on console

An finally we will print the DOM tree on the console with the following code:

```
TransformerFactory tranFactory =
TransformerFactory.newInstance();
Transformer aTransformer =
tranFactory.newTransformer();
Source src = new DOMSource(doc);
Result dest = new
StreamResult(System.out);
aTransformer.transform(src, dest);
```

Here is the full source code of **CreateDomXml.java**

```
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilder;
import
javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import
javax.xml.transform.stream.StreamResult;

class CreateDomXml
{
  public static void main(String[] args)
  {
    try{
    //Create instance of
DocumentBuilderFactory
    DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
    //Get the DocumentBuilder
    DocumentBuilder docBuilder =
factory.newDocumentBuilder();
    //Create blank DOM Document
        Document doc =
docBuilder.newDocument();

    //create the root element
        Element root =
doc.createElement("root");
    //add it to the xml tree
        doc.appendChild(root);

      //create a comment
      Comment comment =
doc.createComment("This is comment");
      //add in the root element
      root.appendChild(comment);

    //create child element
    Element childElement =
doc.createElement("Child");
    //Add the atribute to the child
    childElement.setAttribute("attribute1","The
value of Attribute 1");
    root.appendChild(childElement);

    TransformerFactory tranFactory =
TransformerFactory.newInstance();
     Transformer aTransformer =
tranFactory.newTransformer();

    Source src = new DOMSource(doc);
    Result dest = new
StreamResult(System.out);
```

```
    aTransformer.transform(src, dest);

   }catch(Exception e){
     System.out.println(e.getMessage());
   }
  }
}
```

The given code will generate the following xml code and display on the console.

```
<?xml version=”1.0" encoding=”UTF-8"
standalone=”no”?>
<root>
<!—This is comment—>
<Child attribute1=”The value of Attribute 1"/
>
</root>
```

Download the Program

Lets see another example that helps you to retrieve the elements as well as their corresponding data from the DOM tree.
In this example you need a well-formed XML file that has some data (Emp_Id, Emp_Name and Emp_E-mail in our case).

Here is the XML File (**emp.xml)** to be parsed:

```
<?xml version = “1.0” ?>
<Employee-Detail>
<Employee>
<Emp_Id> E-001 </Emp_Id>
<Emp_Name> Vinod </Emp_Name>
<Emp_E-mail> Vinod1@yahoo.com </
Emp_E-mail>
</Employee>
<Employee>
<Emp_Id> E-002 </Emp_Id>
<Emp_Name> Amit </Emp_Name>
<Emp_E-mail> Amit2@yahoo.com </Emp_E-
mail>
</Employee>
<Employee>
<Emp_Id> E-003 </Emp_Id>
<Emp_Name> Deepak </Emp_Name>
<Emp_E-mail> Deepak3@yahoo.com </
Emp_E-mail>
</Employee>
</Employee-Detail>
```

Develop a java file (**GetDomData.java**) that uses an xml file to parse. Initially the program checks that the given file exists or not by using **exists()** method. It determines that the parsed xml is well formed or not. If you enter a file that doesn’t exist it will show “File not found!”.

Here is a sample code of this program:

```
File file = new File(xmlFile);
      if (file.exists()){
     DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
  DocumentBuilder builder =
factory.newDocumentBuilder();
Document doc = builder.parse(xmlFile);
    System.out.println(xmlFile + “ is well-
formed “);
```

To parse the xml file you need the DoucnemtBuilderFactory and DocumentBuilder. The object of the DocumentBuilder uses **parse** method and determines that the parsed xml is well formed or not. If xml document is will-formed, it will display a message “emp.xml is well-formed!” Otherwise prints “emp.xml isn’t well-formed!”.

Now, lets see the sample code to retrieve the elements from the xml file:

```
      NodeList list =
doc.getElementsByTagName(“*”);
      for (int i=0; i<list.getLength(); i++){
        // Get element
        Element element =
(Element)list.item(i);
        //Source src = new
DOMSource(element);
      System.out.println(element.getNodeName());
    }
```

The doc object helps in create a **NodeList** through the **getElementByTagName()** method. The NodeList helps you in getting the length and Element. For getting the node name you use the **getNodeName()** method.

Now, lets see the sample code to retrieve the data from the elements:

```
//Create transformer
  Transformer tFormer =
TransformerFactory.newInstance().newTransformer();
// Output text type

tFormer.setOutputProperty(OutputKeys.METHOD,
"text");
//Write the document to a file
         Source source = new
DOMSource(doc);
  Result result = new
StreamResult(System.out);
tFormer.transform(source, result);
```

The setOutputProperty method of the Transformer class set the output key as a text to print the data on console. Then the **transform()** method displays the data source and the given destination. This program uses the "System.out" to display the data on the console.

Here is full source code of **GetDomData.java:**

```
import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import
javax.xml.transform.stream.StreamResult;


public class GetDomData{
  static public void main(String[] arg) {
    try{
      BufferedReader bf = new
BufferedReader(new
InputStreamReader(System.in));
      System.out.print("Enter XML file name:
");
      String xmlFile = bf.readLine();
      File file = new File(xmlFile);
      if (file.exists()){
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder =
factory.newDocumentBuilder();
        Document doc = builder.parse(xmlFile);
        System.out.println(xmlFile + " is well-
formed ");
        NodeList list =
doc.getElementsByTagName("*");
      for (int i=0; i<list.getLength(); i++){
        // Get element
        Element element =
(Element)list.item(i);
        //Source src = new
DOMSource(element);
      System.out.println(element.getNodeName());
    }
    //        Create transformer
       Transformer tFormer =
TransformerFactory.newInstance().newTransformer();
//        Output text type

tFormer.setOutputProperty(OutputKeys.METHOD,
"text");
//        Write the document to a file
       Source source = new DOMSource(doc);
       Result result = new
StreamResult(System.out);
       tFormer.transform(source, result);

     }
     else{
       System.out.println("File not found!");
     }
    }
    catch (Exception e){
      System.err.println(e);
      System.exit(0);
    }
  }
}
```

Output of the Program:

```
C:\nisha>javac GetDomData.java

C:\nisha>java GetDomData
Enter XML file name: emp.xml
emp.xml is well-formed
Employee-Detail
Employee
Emp_Id
Emp_Name
Emp_E-mail
Employee
Emp_Id
Emp_Name
```

```
Emp_E-mail
Employee
Emp_Id
Emp_Name
Emp_E-mail

 E-001
 Nisha
 nisha1@yahoo.com

 E-002
 Amit
 amit2@yahoo.com

 E-003
 Deepak
 deepak3@yahoo.com

C:\nisha>
```

Struts 1.1

This tutorial provides you a better understanding to develop a robust application using Jakarta Struts Framework. Before starting we are considering that you have an idea about developing a web application by using JSP, Servlets, JDBC and custom tags. So let's from the Jakarta Struts Framework.

Struts

Struts Frame is nothing but the implementation of Model-View-Controller (MVC) design pattern. Struts is an open source framework and maintained as a part of Apache Jakarta project. Struts Framework suits to develop any size of application. You can down the latest version of struts from http://jakarta.apache.org/. We are using jakarta-struts-1.1 and jakarta-tomcat-5.0.4 for this application. Before going to start lets first take a brief introduction about MVC architecture.

Model-View-Controller (MVC) Architecture

Model-View-Controller architecture separates the application components into three different parts named Model, View and the Controller. Each component of the MVC architecture is independent with the other and also has unique responsibility. Changes made to one component have less or no impact on other component. Here is description about the responsibilities of the components of MVC architecture:

Model: This part of the MVC architecture provides access to the data from the database and also saves the data into the database. Model includes the business logic of the application. Model part also checks the Data send by the user through View before saving it into the database. Data access, Data validation and the data saving logic are also part of Model.

View: View part of the MVC architecture handle the view part of the application by taking the input from the user, dispatching the request to the controller and receiving response from the controller and displaying the result to the user. HTML, JSPs, Custom Tag Libraries and Resources files are used as part of view component.

Controller: Controller part controls the flow of the entire application. It interacts with Model and View and the works as Intermediary between these two components of the MVC architecture. Controller also receives the request from client process that request by executing the appropriate business logic of the Model and then sends the response to the user using the View component. Controller part includes ActionServlet, Action, ActionForm and struts-config.xml as its parts.

Setting Up Development Environment

Before going to start first of all we have to setup the development environment for the application.

JDK Installation:

Download JDK 1.4 or above from sun site. Follow the instruction given in the installation process to install JDK on your machine.

Tomcat Installation:
Download the binary version of Tomcat from the apache site and follow the instruction to install the tomcat on your machine. For this application we have downloaded jakarta-tomcat-5.0.4. After successfully completion of the installation process test your installation. To test the installation, go to your installation **directory/bin** and issue startup command to run the server. Open the browser and type http://localhost:8080/ . If it is successfully installed it should display the welcome page, If not consult tomcat documentation before going further.

Installing Struts Application:
Download any version of Struts1 from the site of Struts http://jakarta.apache.org/struts. Extract file into the desired directory and then copy **struts-blank.war**, **struts documentation.war** and **struts-example.war** from "**jakarta-struts-1.1\webapps**" directtory into "**jakarta-tomcat-5.0.4\webapps**" directory.

struts-blank.war is a blank struts application which is useful in creating struts application from scratch. We are using this file to create our

own web application.

struts-documentation.war includes important documents and API for the struts application development.

struts-example.war is simple MailReader Demonstration Application.

Developing First Struts Application

Rename **struts-blank.war** to **StrutsApplication.war** from **jakarta-tomcat-5.0.4\webapps** and copy it to the "**jakarta-tomcat-5.0.4\webapps**" directory. Tomcat automatically extracts the file and loads the application.

Copy the source files (LookupDispatch_Action.java and MappingDispatch_Action.java) into the source directory (src directory), jsp files MappingDispatchAction.jsp and MappingDispatchActionSave.jsp into the pages directory, index.jsp file in the StrutsApplication directory parallel to the pages and WEB-INF directory, struts-config.xml, web.xml, struts-bean.tld, and struts-html.tld parallel to the lib and the classes directory in the WEB-INF directory, and finally copy the servlet-api.jar and struts.jar files into the lib directory and then compile the whole application, start the tomcat server, open the browser and enter the url http://localhost:8080/StrutsApplication/ and then press enter, if everything is ok then the welcome page (that is index.jsp in case of our application) will be displayed. Here are the different files used in our application.

LookupDispatch_Action.java

```
package roseindia.net;
import java.io.*;
import java.util.*;
import javax.servlet.http.HttpServletRequest;
import
javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import
org.apache.struts.actions.LookupDispatchAction;
import org.apache.struts.action.ActionForm;
import
org.apache.struts.action.ActionForward;
import
org.apache.struts.action.ActionMapping;
public class LookupDispatch_Action extends
LookupDispatchAction {
protected Map getKeyMethodMap(){
Map map = new HashMap();
  map.put("roseindia.net.add","add");
map.put("roseindia.net.edit","edit");
map.put("roseindia.net.search","search");
map.put("roseindia.net.save","save");
          return map;
}
  public ActionForward add(ActionMapping
mapping, ActionForm form,
  HttpServletRequest request,
HttpServletResponse response) throws
Exception{
    System.out.println("You are in add
function.");
                    return
mapping.findForward("add");
  }
  public ActionForward edit(ActionMapping
mapping, ActionForm form,
  HttpServletRequest request,
HttpServletResponse response) throws
Exception{
System.out.println("You are in edit
function.");
return mapping.findForward("edit");
  }
   public ActionForward search(ActionMapping
mapping, ActionForm form,
HttpServletRequest request,
HttpServletResponse response) throws
Exception{
    System.out.println("You are in search
function");
                    return
mapping.findForward("search");
}

public ActionForward save(ActionMapping
mapping, ActionForm form,
          HttpServletRequest request,
HttpServletResponse response) throws
Exception{
    System.out.println("You are in save
function");
                    return
```

```
mapping.findForward("save");
  }
}
```

MappingDispatch_Action.java

```
package roseindia.net;
import java.io.*;
import java.util.*;
import javax.servlet.http.HttpServletRequest;
import
javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import
org.apache.struts.actions.MappingDispatchAction;
import org.apache.struts.action.ActionForm;
import
org.apache.struts.action.ActionForward;
import
org.apache.struts.action.ActionMapping;
public class MappingDispatch_Action extends
MappingDispatchAction {
     public ActionForward add(ActionMapping
mapping, ActionForm form,
HttpServletRequest request,
HttpServletResponse response) throws
Exception{
    System.out.println("You are in add
function.");
                    return
mapping.findForward("add");
  }
    public ActionForward edit(ActionMapping
mapping, ActionForm form,
HttpServletRequest request,
HttpServletResponse response) throws
Exception{
    System.out.println("You are in edit
function.");
                    return
mapping.findForward("edit");
  }
   public ActionForward search(ActionMapping
mapping, ActionForm form,
HttpServletRequest request,
HttpServletResponse response) throws
Exception{
System.out.println("You are in search
function");
return mapping.findForward("search");
  }
  public ActionForward save(ActionMapping
mapping, ActionForm form,
HttpServletRequest request,
HttpServletResponse response) throws
Exception{
System.out.println("You are in save
function");
return mapping.findForward("save");
  }
}
```

MappingDispatchAction.jsp

```
<%@ taglib uri="/WEB-INF/struts-bean.tld"
prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html"%>
<html:html locale="true">
<HEAD>
 <BODY>
<p><html:link page="/
MappingDispatchAction.do?parameter=add">Call
Add Section</html:link></p>
<p><html:link page="/
MappingDispatchAction.do?parameter=edit">Call
Edit Section</html:link></p>
<p><html:link page="/
MappingDispatchAction.do?parameter=search">Call
Search Section</html:link></p>
<p><html:link page="/
MappingDispatchAction.do?parameter=save">Call
Save Section</html:link></p>
</html:html>
```

MappingDispatchActionSave.jsp

```
<html>
<head>
<title>Success Message</title>
</head>
<body>
<p align="center"><font size="5"
color="#000080">Welcome to save Page</
font></p>
</body>
</html>
```

index.jsp

```
<%@ taglib uri="/tags/struts-html"
prefix="html" %>
<html:html locale="true">
<head>
```

```
<html:base/>
</head>
<body>
<p align="center"><font size="5"
color="#800000">Welcome to the Simple
Struts Application</font></p>
<div align="center">
  <center>
  <table border="1" cellspacing="1"
width="400">
 <li>
<html:link page="/pages/
MappingDispatchAction.jsp">Demo of a
Simple Struts Application</html:link>
</li>
</table>
</center>
</div>
</body>
</html:html>
```

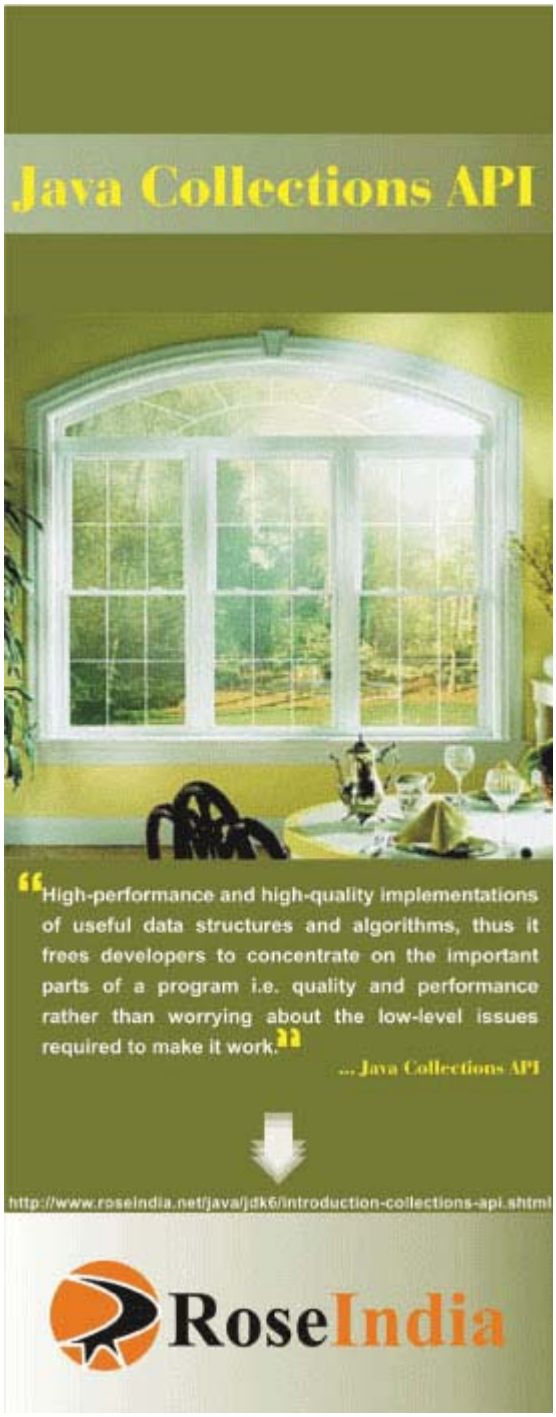
struts-config.xml

```
<?xml version="1.0" encoding="ISO-8859-1"
?>
<!DOCTYPE struts-config PUBLIC "-//Apache
Software Foundation//DTD Struts
Configuration 1.2//EN""http://
jakarta.apache.org/struts/dtds/struts-
config_1_2.dtd">
<struts-config>
    <action-mappings>
<action  path="/LookupDispatchAction"
type="roseindia.net.LookupDispatch_Action"
parameter="parameter"
input="/pages/LookupDispatchAction.jsp"
scope="request"
validate="false">
          <forward name="add" path="/pages/
LookupDispatchActionAdd.jsp" />    <forward
name="edit" path="/pages/
LookupDispatchActionEdit.jsp" />
<forward name="search" path="/pages/
LookupDispatchActionSearch.jsp"/>
<forward name="save" path="/pages/
LookupDispatchActionSave.jsp" />
       </action>
<action
path="/MappingDispatchAction"
type="roseindia.net.MappingDispatch_Action"
parameter="add"
input="/pages/MappingDispatchAction.jsp"
scope="request"
validate="false">
          <forward name="add" path="/pages/
MappingDispatchActionAdd.jsp" />
  </action>
  <action
          path="/MappingDispatchAction"
      type="roseindia.net.MappingDispatch_Action"
          parameter="edit"
          input="/pages/
MappingDispatchAction.jsp"
          scope="request"
          validate="false">
  <forward name="edit" path="/pages/
MappingDispatchActionEdit.jsp" />
  </action>
  <action
          path="/MappingDispatchAction"
      type="roseindia.net.MappingDispatch_Action"
          parameter="search"
          input="/pages/
MappingDispatchAction.jsp"
          scope="request"
          validate="false">
          <forward name="search" path="/
pages/MappingDispatchActionSearch.jsp"/>
  </action>
  <action
          path="/MappingDispatchAction"
      type="roseindia.net.MappingDispatch_Action"
          parameter="save"
          input="/pages/
MappingDispatchAction.jsp"
          scope="request"
          validate="false">
         <forward name="save" path="/pages/
MappingDispatchActionSave.jsp" />
  </action>
   </action-mappings>
</struts-config>
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-
1"?>
<!DOCTYPE web-app PUBLIC "-//Sun
Microsystems, Inc.//DTD Web Application 2.2/
/EN" "http://java.sun.com/j2ee/dtds/web-
app_2_2.dtd">
<web-app>
```

```
  <display-name>Struts Blank Application</
display-name>
  <!— Standard Action Servlet Configuration
(with debugging) —>
  <servlet>
   <servlet-name>action</servlet-name>
    <servlet-
class>org.apache.struts.action.ActionServlet</
servlet-class>
    <init-param>
     <param-name>config</param-name>
      <param-value>/WEB-INF/struts-
config.xml</param-value>
    </init-param>
    <init-param>
     <param-name>debug</param-name>
     <param-value>2</param-value>
    </init-param>
    <init-param>
     <param-name>detail</param-name>
     <param-value>2</param-value>
    </init-param>
     <load-on-startup>2</load-on-startup>
  </servlet>
  <!— Standard Action Servlet Mapping —>
  <servlet-mapping>
   <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <!— The Usual Welcome File List —>
  <welcome-file-list>
   <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <!— Struts Tag Library Descriptors —>
  <taglib>
    <taglib-uri>/tags/struts-html</taglib-uri>
     <taglib-location>/WEB-INF/struts-
html.tld</taglib-location>
  </taglib>
</web-app>
```



Struts 2 Non-form Tags (UItags)

Apache Struts is an open-source framework used to develop Java web applications. In this section, struts 2 non-form tags (UItags) will be discussed. Just download the zip file "struts2nonformuitags.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/struts2nonformuitags to the address bar. You can examine the result of each tag from the index page.

1. Action Error and Action Message Tags Example

The actionerror tag is a UI tag that renders action errors (in the jsp pages.) if they exist while the actionmessage tag renders action messages if they exist.

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

struts.xml

```
<action name="actionerrorTag">
  <result>/pages/nonformTags/login.jsp</
result>
</action>
<action name="login"
class="net.javajazzup.CheckValidUser">
    <result name="input">/pages/
nonformTags/login.jsp</result>
    <result name="error">/pages/
nonformTags/error.jsp</result>
    <result>/pages/nonformTags/
validuser.jsp</result>
</action>
```

Create an action class that uses methods addActionMessage(String) and addActionError(String) within the execute() method. The addActionMessage(String) will print the passed string on the success jsp page while the addActionError(String) will print the passed string on the error jsp page.

CheckValidUser.java

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

public class CheckValidUser extends
ActionSupport {
 private String username = null;
 private String password = null;
 public String execute() throws Exception{
  if ((getUsername().equals("javajazzup"))
&&
   (getPassword().equals("javajazzup"))){

    addActionMessage("Valid User");
     return SUCCESS;
   }
   else{
    addActionError("Invalid User");
     return ERROR;
   }
 }
```

```
 //Set and get the user name
 public void setUsername(String name){
   username = name;
 }
 public String getUsername(){
   return username;
 }
 //set and get the password
 public void setPassword(String pass){
   password = pass;
 }
 public String getPassword(){
   return password;
 }
}
```

Create a login jsp page as shown:

login.jsp

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

<html>
 <head>
  <title>Actionerror Tag Example</title>
 <body>
 <s:form action="login" method="POST">
 <s:textfield label="User Name"
name="username" size="20" maxlength="10"
/>
 <s:password label="Password"
name="password" size="20" maxlength="10"
/>
 <s:submit value="Submit" />
 </s:form>

 </body>
</html>
```

Create a jsp page that will display your error message (when fails to logged-in) using the empty <s:actionerror /> tag as shown:

error.jsp

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

<html>
 <head>
  <title></title>
 <body>
 <h1>
 <s:actionerror />
 </h1>
  <a href="/struts2nonformuitags/
javajazzup/actionerrorTag.action">Go Back</
a>
 </body>
</html>
```

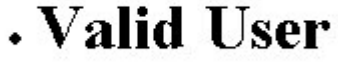
Create a jsp page that will display a message (when successfully logged-in) using the empty <s:actionmessage /> tag as shown:

validuser.jsp

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

<html>
 <head>
  <title>Actionerror Tag Example</title>
 <body>
 <h1>
 <s:actionmessage />
 </h1>
 </body>
</html>
```

You will see the output of the login.jsp as shown below. Put correct user name and password.

You will get output:



Enter the wrong user name or password in the login page.

You will get the following output:

2. Div (Ajax Tag) tag Example

The div tag is an Ajax component that is used with Ajax that refreshes the content of a particular section without refreshing the entire page. The div tag when used with Ajax refreshes the content of a particular section without refreshing the entire page. Html tag created by Ajax div tag includes it's content and is used to obtain it's content through a remote XMLHttpRequest call through the dojo framework.

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

struts.xml

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

Create a jsp using the tag <s:div>.
div.jsp

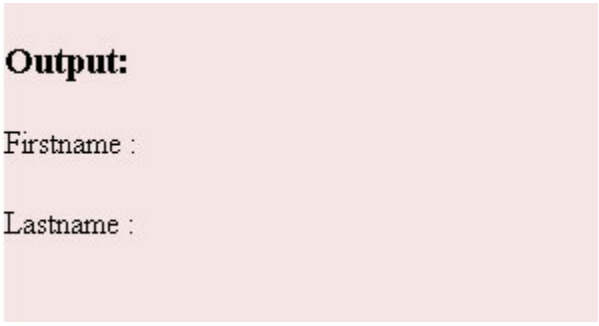
```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
   <head>
      <title>Enter first and last name</title>
   <s:head theme="ajax" debug="false"/>
   </head>
   <body>
   <s:url id="test" value="/pages/
nonformTags/mask.jsp" />
   <s:div
    id="one"
    theme="ajax"
    href="%{test}">
   </s:div>
   </body>
</html>
```

mask.jsp

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

<html>
   <head>
      <title>Enter first and last name</title>
      <s:head theme="ajax" debug="false"/>

   </head>
   <body>
   <s:div id="maskValue" >
   <div style="position:absolute;top:10;
left:20; width:300;
height:175;background-color:#E5E5E5;">
   <h3>Enter first and last name:</h3>
    <s:form theme="ajax" action="doMask">
      <s:textfield name="firstname"
label="Firstname" />
      <s:textfield name="lastname"
label="Lastname" />
      <s:submit value="Submit"
theme="ajax" targets="maskValue" />
    </s:form>
   </div>
   <br>

   <div id="8"
style="position:absolute;top:10; left:350;
width:300; height:160;background-
color:#E5E5E5;">
    <h3>Output: </h3>
    Firstname : <s:property
value="firstname" />
    <br><br>
    Lastname :  <s:property
value="lastname" />
    </div>
```

```
   </s:div>
   </body>
</html>
```

Output:



3. Fielderror Tag (Non-Form UI Tags) Example

The fielderror tag is a UI tag that renders field errors if they exist.

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

struts.xml

```
<action name="fieldError">          <result>/
pages/nonformTags/loginFielderrorTag.jsp</
result>
</action>
<action name="checkUser"
class="net.javajazzup.CheckField">
          <result name="input">/pages/
nonformTags/loginFielderrorTag.jsp</result>
  <result name="error">/pages/nonformTags/
fielderrorTag.jsp</result>
  <result>/pages/nonformTags/
validuser.jsp</result>
</action>
```

Develop an action class using addFieldError(String fieldName, String errorMessage) method. This method adds an error message for a given field to the corresponding jsp page.

CheckField.java

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

public class CheckField extends ActionSupport
{
  private String username = null;
  private String password = null;
  public String execute() throws Exception{
   if ((getUsername().equals("javajazzup"))
&&
        (getPassword().equals("javajazzup"))){
     addActionMessage("Valid User!");
      return SUCCESS;
    }
   if(!(getUsername().equals("javajazzup")))
   addFieldError("username","Invalid
username!");

   if(!(getPassword().equals("javajazzup")))
   addFieldError("password","Invalid
password!");

    return ERROR;
  }
  //Set and get the user name
  public void setUsername(String username){
    this.username = username;
  }
  public String getUsername(){
    return username;
  }
  //set and get the password
  public void setPassword(String pass){
    password = pass;
  }
  public String getPassword(){
```

```
    return password;
  }
}
```

Create a login jsp page as shown:

loginFielderrorTag.jsp

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

Create a jsp page that will display your field error messages (when fails to logged-in) using the empty <s:fielderror/> tag as shown:

fielderrorTag.jsp

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

<html>
  <head>
    <title>Fielderror Tag Example!</title>
  <body>
  <h1><s:fielderror /></h1>
   <a href="/struts2nonformuitags/
javajazzup/fieldError.action">Go Back</a>
  </body>
</html>
```

Create a jsp page that will display your messages (when succeed to logged-in) using the empty <s:actionmessage /> tag as shown:
validuser.jsp

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

<html>
  <head>
    <title>Actionerror Tag Example!</title>
  <body>
  <h1>
  <s:actionmessage />
  </h1>
  </body>
</html>
```

Enter the correct user name and wrong password in the login page. You will see the output of the loginFielderrorTag.jsp as shown below:

- **Invalid password**

Go Back

Enter the wrong user name and correct password in the login page. You will get the following output:

- **Invalid username**

Go Back

Enter incorrect values in both fields of the login page. You will get the following output:

- **Invalid username**
- **Invalid password**

Go Back

Enter correct values in both fields of the login page. You will get the following output:

- **Valid User**

4. TabbedPanel (Ajax Tag) Example

This is an Ajax component, where each tab is either a local content or a remote content (refreshed each time when user selects that

tab).

To use tabbedPanel tag, the head tag must be included on the jsp page and must be configured for performance or debugging purposes. However, If you want to use the cookie feature then you must provide a unique id for the tabbedpanel component. This is used for identifying the name of component that is stored by the cookie.

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

struts.xml

```
<action name="tabbedPanel">
  <result>/pages/nonformTags/
tabbedpanel.jsp</result>
</action>
```

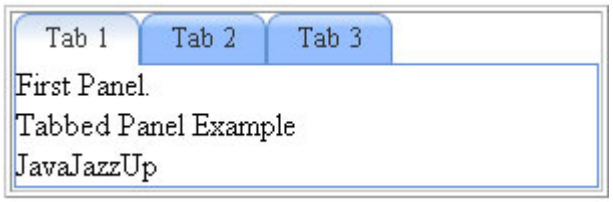
Create a jsp using the tag <s:tabbedPanel>. This tag is used for creating the tabs.

tabbedpanel.jsp

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

<html>
 <head>
  <s:head theme="ajax" debug="true"/>
 </head>
 <body>
 <table border="1" width="30%">
  <tr>
  <td width="100%">

  <s:tabbedPanel id="test" >

   <s:div id="one" label="Tab 1"
theme="ajax" labelposition="top" >
      First Panel.<br>
      Tabbed Panel Example<br>
   JavaJazzUp
    </s:div>

   <s:div id="two" label="Tab 2"
theme="ajax">
      Second Panel.<br>
      JavaJazzUp
    </s:div>

   <s:div id="three" label="Tab 3"
theme="ajax">
      Third Panel.<br>
      JavaJazzUp
    </s:div>

  </s:tabbedPanel>

  </td>
  </tr>
 </table>
 </body>
</html>
```

Output:

When you run the above example, you get the output as:



5. tree and treenode (Ajax Tag) tags Example

In this section, you will learn about the tree and treenode tags. These both work with the Ajax support.

tree: This is a tree widget with AJAX support. Normally this tag uses the "id" attribute. The "id" attribute is required if the "selectedNotifyTopic" or the "href" attribute is going to be used.

treenode: This is a tree node which renders a tree node within a tree widget with AJAX support. The following of the two combinations are used depending on the requirement like the tree is needed to be constructed dynamically

or statically.

Dynamically

- id - This is an id of tree node.
- title - This is as like a label to be displayed for the tree node

Statically

- rootNode - This is the parent node where the tree is derived form.
- nodeIdProperty - This is the current tree node's id.
- nodeTitleProperty - This is the current tree node's title.
- childCollectionProperty - This is the current tree node's children.

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

struts.xml

```
<action name="TreeNode">
      <result>/pages/nonformTags/
treenode.jsp</result>
</action>
```

Create a jsp using the tag <s:tree>. This tag is used for rendering a tree widget.

Similarly the tag <s:treenode> renders a tree node with a label attached to the created tree widget eg..

<s:treenode theme="ajax" id="subchild2" label="subchild2" /> tag attaches a new node to the created tree with a tree node having id="subchild2" and a label="subchild2".

treenode.jsp

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

<html>
 <head>
  <s:head theme="ajax" debug="true"/>
 </head>
 <body>
  <s:tree theme="ajax" id="root"
label="Root">
    <s:treenode theme="ajax" id="child1"
label="<b>Child 1</b>" />
      <s:treenode theme="ajax"
id="subchild1" label="SubChild 1">
       <s:treenode theme="ajax"
id="subchild2" label="SubChild 2" />
       <s:treenode theme="ajax"
id="subchild3" label="SubChild 3" />
      </s:treenode>
    <s:treenode theme="ajax" id="child2"
label="<b>child 2</b>" />
   </s:tree>
 </body>
</html>
```

Output:

When you run the above example, you get:

Design Pattern

1. Template Design Pattern

These types of design patterns are used as templates. These design patterns are used in such conditions when we need a parent class having one or more methods to be implemented by their child classes. This design pattern introduces an idea of defining an algorithm in a class and leaving some of the methods to be implemented by their subclasses.

This design pattern is used to develop similar kind of operations template, reusing the common behavior to simplify code, algorithm related improvement, from many generalized to specialized operations.

Lets take an example of loan application, this application may have several steps to complete its processing.

Here we are illustrating several steps to complete:

- Bank balance history of a client's check.
- Credit score of the client's check taken from three different companies.
- Other loan information of client's check.
- Stock holding value of a client's check.
- Future income potential of that client's check.

In all of the above situations we can use a template method hold the steps of the process together without deliberating the actual implementation of the process's steps together without considering the real implementation in the subclass.

```
abstract class CheckVariousdtls {

   public abstract void Bankdetails();
   public abstract void Creditdetails();
   public abstract void Loandetails();
   public abstract void Stockdetails();
   public abstract void Incomedetails();

   public void checkdetails() {
      Bankdetails();
      Creditdetails();
      Loandetails();
      Stockdetails();
      Incomedetails();
   }
}

class LoanApplication extends
CheckVariousdtls {
   private String name;

   public LoanApplication(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }

   public void Bankdetails() {
      System.out.println("Checking bank
details...");
   }

   public void Creditdetails() {
      System.out.println("Checking credit
details...");
   }

   public void Loandetails() {
      System.out.println("Checking other loan
details...");
   }

   public void Stockdetails() {
      System.out.println("Checking stock
values details...");
   }

   public void Incomedetails() {
      System.out.println("Checking family
income details...");
   }

}

class TestTemplateApplication {
   public static void main(String[] args) {

      LoanApplication Clientdtls = new
LoanApplication("Amit");
```

```
        System.out.println("\nChecking " +
Clientdtls.getName()+ " (client) loan details.
");
        Clientdtls.checkdetails();

        LoanApplication Clientloandtls = new
LoanApplication("Aqueel");
        System.out.println("\nChecking " +
Clientloandtls.getName()+ " (client) loan
details. ");
        Clientloandtls.checkdetails();
    }
}
```

Best example of template design pattern is method overloading and method overriding. For example,

The example illustrated below has an add() method that is a template method. This add() method may take any primitive type numerical value and we can type cast the result according to our requirement.

```
abstract class AddNumbers {
   public abstract double addnumbers(double
d1, double d2);
}
class AddAnyTypeOfNumber extends
AddNumbers{
    public double addnumbers(double d1,
double d2) {
        return d1 + d2;
    }
}
class TestApplication {
   public static void main(String[] args) {
       byte b1 = 6, b2 = 2;
      short s1 = 4, s2 = 6;
       int i1 = 1, i2 = 8;
       long l1 = 5, l2 = 9;
       float f1 = 14.7f, f2 = 37.9f;
      double d1 = 12.7, d2 = 11.2;


       AddAnyTypeOfNumber
addanytypeNumber = new
AddAnyTypeOfNumber();
```

```
System.out.println(addanytype
Number.addnumbers(d1,d2));
System.out.println((float)addanytype
Number.addnumbers(f1,f2));
System.out.println((long)addanytype
Number.addnumbers(l1,l2));
System.out.println((int)addanytype
Number.addnumbers(i1,i2));
System.out.println((short)addanytype
Number.addnumbers(s1,s2));
System.out.println((byte)addanytype
Number.addnumbers(b1,b2));
   }
}
```

Visitor Design Pattern

2. Visitor Design Pattern

The design pattern provides additional functionality to a class. The Visitor pattern allows us to create an external class to act on data in other classes. This is useful in those conditions if a fair number of instances are available of small number of classes and we have to perform some operations on all or on most of those classes.

Lets take an example that demonstrates that how the classes implement the Visitor interface. In the following example we are taking the two interfaces one is Visitor interface and the other is Colddrink interface and then we implement to these interfaces into our classes.

import java.util.*;

```
interface Visitor {
   public void visit(Colddrink cld_drnk);
}
interface Colddrink {
   public String Order();
}
class Pepsi implements Colddrink {
   final String name = "Pepsi";
   public String Order() {
      return name;
   }
}
class CocaCola implements Colddrink {
   final String name = "Coke";
   public String Order() {
      return name;
   }
}

class Pickup implements Visitor {
   private String name;
   private final String method = "Pick and
Go";
   public void visit(Colddrink cld_drnk) {
      name = cld_drnk.Order();
   }

   public String toString() {
       return name + " " + method;
   }
}
class Drinkthere implements Visitor {
   private String name;
   private final String method = "Take the
drink over there";

   public void visit(Colddrink cld_drnk) {
      name = cld_drnk.Order();
   }

   public String toString() {
      return name + " " + method;
   }
}
class GetByDelivery implements Visitor {
   private String name;
   private final String method = "Get it by
delivery";

   public void visit(Colddrink cld_drnk) {
      name = cld_drnk.Order();
   }

   public String toString() {
       return name + " " + method;
   }
}
class Drink {
   public Colddrink getDrink() {
      switch ((int)(Math.random()*3)){
         case 0: return new Pepsi();
         case 1: return new CocaCola();
         default: return null;
       }
   }
   public Visitor howto() {
      switch ((int)(Math.random()*3)){
         case 0: return new Pickup();
         case 1: return new Drinkthere();
         case 2: return new GetByDelivery();
         default: return null;
       }
   }
}
class TestVisitors {

   public static void main(String[] args) {
      List colddrinkList = new ArrayList();
         colddrinkList.add(new Pepsi());
         colddrinkList.add(new CocaCola());

      Iterator it = colddrinkList.iterator();
```

```
        System.out.println("How many Cold
drink Shop are there in this area?");
        while (it.hasNext()) {
      System.out.println(((Colddrink)it.next()).Order());
        }
        Drink drink = new Drink();
        Colddrink cld_drnk = drink.getDrink();
         Visitor vstr = drink.howto();
         vstr.visit(cld_drnk);
         System.out.println("\n\nChoose the
shop to take the drink?");
         System.out.println(vstr);
    }
}
```

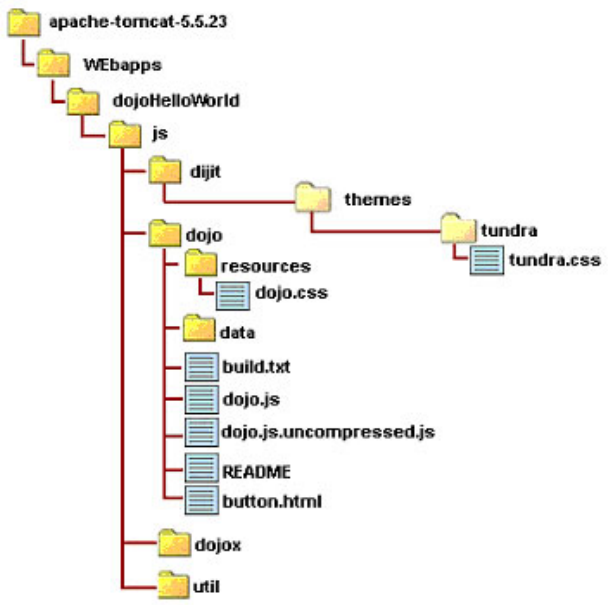
Dojo Tutorial

Dojo Tutorial

Dojo: Dojo is an Open Source JavaScript toolkit libraries that provides a simple API(Application Programming Interface) for building the serious applications in less time. The main functionality of Dojo is to make HTTP requests and receive their responses. It also provides packages for string manipulation, DOM manipulation, drag-and-drop support and DS (Data Structures) such as lists, queues and stacks. Dojo applications are used where the JavaScript and browsers don't work far enough, in which place the dojo application gives you the powerful, portable, lightweight and tested tools for creating a dynamic interfaces.

Dojo Directory Structure:

Whenever you use Dojo then you follow the following directory structure and set up the files of the specified location.



Hello world Example

Here, you will learn to create a "Hello World" example in Dojo. Before creating any examples or applications you must follow the directory structure.

Create a Button:

The following example we are going to create a button "Hello World!". To create a button in dojo you need to a Button Widget that contains three visual states as: mouseOut, mouseOver and mouseDown. To follow the following steps for creating a dojo button widget:

```
<script type="text/javascript">
        // Load Dojo's code relating to
widget managing functions
        dojo.require("dojo.widget.*");

        // Load Dojo's code relating to the
Button widget
        dojo.require("dojo.widget.Button");
</script>
```

dojo.require("dojo.widget.*"): It instructs you to include the dojo widget (Not load all the widgets) for managing functions.

dojo.require("dojo.widget.Button"): This line instructs you to load the Dojo button widget. If you don't include this line, the markup code for the button would not be evaluated by Dojo upon loading, resulting in a plain HTML button instead of what you expect.

Insert the following code into the HTML body:

```
<button dojoType="Button"
widgetId="helloButton"
onClick="helloPressed();">Hello World!</
button>
```

The key attribute of this HTML element to notice is the dojoType attribute. This is responsible for instructing Dojo on how to process the element when the page is loading. In this case you will use a button element for the button that is used to input element - Dojo will work with either as long as the dojoType attribute is present.

widgetId="helloButton": This is replaced with id="helloButton" without the functionality being affected - Dojo's widget system is smart

enough to convert regular id attributes to widgetId's if no widgetId' attribute is explicitly named.

Connecting an Event to the Widget
When you click the command button then it doing something? We specify an **onClick** event handler for the given command button.

dojo.require("dojo.event.*");
Above code we use "dojo.event.*" that includes all events functionality of Dojo (But not all widgets).

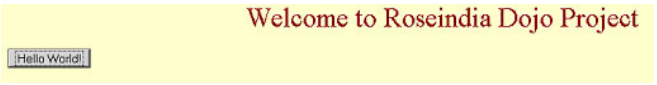
Following function that will called by the button when we clicked. After clicking the "helloPressed" method is called and it displays an alert message like: "Click on the Hello World Button".

```
    function helloPressed()
  {
  alert('Click on the Hello World Button');
  }
```

Here is the code of program:

```
<html>

<head>
<title>button</title>
  <script type="text/javascript">
    dojo.require("dojo.event.*");
    dojo.require("dojo.widget.*");
    dojo.require("dojo.widget.Button");

    function helloPressed()
    {
    alert('Click on the Hello World Button');
    }

    function init()
    {
    var helloButton =
dojo.widget.byId('helloButton');
    dojo.event.connect(helloButton, 'onClick',
'helloPressed')
    }

  dojo.addOnLoad(init);
  </script>
</head>

<body bgcolor="#FFFFCC">

<p align="center"><font size="6"
color="#800000">Welcome to Roseindia
Dojo Project</font></p>

<button dojoType="Button"
widgetId="helloButton"
onClick="helloPressed();">Hello World!</
button>
<br>

</body>

</html>
```

Output of program:



Tool tips Example

Tool tips: This is a GUI (Graphical User Interface) element that is used in conjunction with a cursor and usually a mouser pointer.

If the mouse cursor over an item, without clicking a mouse it appears a small box with supplementary information regarding the items.

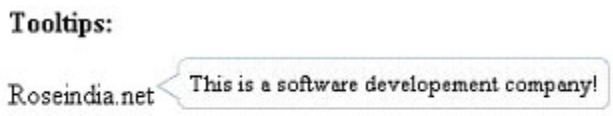
Here is the code of program:

```
<html>
<head>
<title>Tooltip Demo</title>

  <style type="text/css">
    @import "../resources/dojo.css";
```

```
    @import "../dijit/themes/tundra/
tundra.css";

   </style>

   <script type="text/javascript"
src="dojo.xd.js" djConfig="parseOnLoad:
true"></script>

   <script type="text/javascript">
     dojo.require("dojo.parser");
     dojo.require("dijit.Tooltip");
   </script>

</head>
<body class="tundra">
<b>Tooltips:</b> <br><br>
      <span id="site1">Roseindia.net</span>
      <div dojoType="dijit.Tooltip"
connectId="site1"
label="This is a software developement
company!">
      </div><br><br><br>
  <span id="site2">Newstrackindia.com</
span>
      <div dojoType="dijit.Tooltip"
connectId="site2" label="This is a news
publishing site!">
      </div>

</body>
</html>
```

Output of the program:

Whenever your mouse pointer goes on the Roseindia.net then you get:



Whenever your mouse pointer goes on the Newstrackindia.com then you get:

Inline DateTextBox

Here,you will learn about the dojo inline DateTextBox and how to create a inline DateTextBox and how to make its editable.
The following code is the InlineEditBox that edits date of **dijit.form.DateTextBox** save it automatically. The inner textarea tag is the Textarea widget. When a user run this code then they see the paragraph of rich text. If user clicks the text, the plain text appears in paragraph. If you want to change the value then click the date text and select the appears date.

The InlineEditBox has methods **get/setDisplayedValue**, inline. The following code shows the DateTextBox that makes inline in HTML.

Here is the code of program:

```
<html>

<head>
 <title>InlineEdit Date Demo</title>

  <style type="text/css">
     @import "../resources/dojo.css";
    @import "../dijit/themes/tundra/
tundra.css";

  </style>

    <script type="text/javascript"
src="dojo.xd.js" djConfig="parseOnLoad:
true"></script>

    <script type="text/javascript">
      dojo.require("dojo.parser");
     dojo.require("dijit.form.InlineEditBox");
      dojo.require("dijit.form.DateTextBox");
    </script>
```

```
</head>
  <body class="tundra">
    <p id="backgroundArea"
dojoType="dijit.form.InlineEditBox" >
    <input name="date" value="2005-12-30"
dojoType="dijit.form.DateTextBox"
            constraints={datePattern:'MM/dd/
yy'}
            lang="en-us"
            required="true"
            promptMessage="mm/dd/yy"
           invalidMessage="Invalid date. Please
use mm/dd/yy format.">
  </body>
</html>
```

Output of the program:

12/30/05

When you click the following date then you get the following and select any date and it automatically save:

Dojo Tree

In this section, you will learn about the tree and how to create a tree in dojo.

Tree : The tree is a GUI that helps to lists the hierarchical lists. The tree widget is a simple but the real power comes in the data. It represents the hierarchical structure of tree. Data is fed by the powerful **dojo.data** API.

There are following steps for creating Dojo trees :

- Create a rooted or rootless trees (forests)
- Nest, each branch is independently expandible
- Different icons for different leaf or branch classes
- Tree data are stored in any **dojo.data** implementing API.
- Events fire when users clicked on it.
- Add, remove or disable nodes of tree.

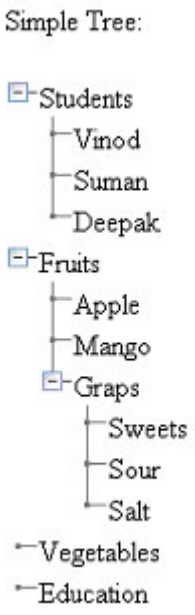
Here is the code of Program:

```
<html>
<title>Tree</title>
<head>

    <style type="text/css">
      @import "../resources/dojo.css";
     @import "../dijit/themes/tundra/
tundra.css";
    </style>

    <script type="text/javascript"
src="dojo.xd.js" djConfig="parseOnLoad:
true"></script>

    <script>
     dojo.require("dojo.data.ItemFileReadStore");
       dojo.require("dijit.Tree");
        dojo.require("dojo.parser");
   </script>

</head>
<body class="tundra">
Simple Tree:<br><br>
        <div
dojoType="dojo.data.ItemFileReadStore"
            url="tree.txt" jsid="popStore" />
        <div dojoType="dijit.Tree"
store="popStore" labelAttr="sname"
label="Tree"></div>
</body>
</html>
```

tree.txt: In JSON format.

```
{ label: 'name',
identifier: 'name',
items: [
{ name:'Students', type:'cat',
children: [
{ name:'Vinod', type:'st' },
{ name:'Suman', type:'st' },
{ name:'Deepak', type:'st' }
]

},
{ name:'Fruits', type: 'cat',
children: [
{ name:'Apple', type:'fr' },
{ name:'Mango', type:'fr' },
{ name:'Graps', type:'fr',
children: [
{ name:'Sweets', type:'gr' },
{ name:'Sour', type:'gr' },
{ name:'Salt', type:'gr' }
]

}
]

},
{ name:'Vegetables', type: 'cat'},
{ name:'Education', type:'cat'}
]
}
```

Output of the Program:



Hibernate Query Language

In the previous issue of Javajazzup you learned about Hibernate Query Language and its different kind of clauses. Lets quickly focus on the overview of HQL.

Introduction to Hibernate Query Language
Hibernate Query Language or HQL for short is extremely powerful query language. HQL is much like SQL and are case-insensitive, except for the names of the Java Classes and properties. HQL has its own object-oriented query language and supports native SQL. It automatically generates the sql query and executes it against underlying database.
Hibernate Query Language is extremely powerful and it supports Polymorphism, Associations, Much less verbose than SQL. It uses Classes and properties instead of tables and columns.

Hibernate uses the following ways to retrieve objects from the database:

- Hibernate Query Language (**HQL**)
- Query By Criteria (**QBC**) and Query BY Example (**QBE**) using **Criteria** API
- Native **SQL** queries

Lets discuss about the Hibernate Criteria APIs in brief.

Introduction to Hibernate Criteria APIs
Hibernate Criteria API is a powerful and elegant alternative to traditional HQL. It is mostly used in case of complex multi criteria search screens, where HQL is not very effective. It provides a well-designed way of building dynamic queries on Hibernate-persisted databases.

The interfaces of the Criteria Query API represent an aspect of the relational approach. There are five core APIs that are commonly used. These are:

1. Criteria
2. Criterion
3. Restrictions
4. Projection
5. Order

1. Criteria

The interface **org.hibernate.Criteria** is used to create the criterion for the search. It is a simplified API for retrieving entities by composing **Criterion** objects. This is a very convenient approach for functionality like "search" screens where there is a number of conditions or fields to be placed upon the result set.

Criteria Interface provides the following methods:

Method	**Description**
add(Criterion criterion)	Adds a Criterion to constrain the results to be retrieved
addOrder(Order order)	Add an Order to the result set.
createAlias(String associationPath, String alias)	Join an association , assigning an alias to the joined entity
createCriteria(String associationPath)	This method is used to create new Criteria, "rooted" at the associated entity.
setFetchSize(int fetchSize)	Set a fetch size for the underlying JDBC query.
setFirstResult(int first Result)	Set the first result to be retrieved.
setMaxResults(int maxResults)	Set a limit upon the number of objects to be retrieved.
uniqueResult()	Convenience method to return a single instance that matches the query, or null if the query returns no results.

Creating a Criteria instance:

To get a reference to the Criteria interface, use the **createCriteria()** method by invoking the **Session** class. This method takes the name of the ORM class on which the query has to be executed. In essence the Session acts as a factory for the Criteria class. The statement for it would be written as:

Criteria criteria=
session.createCriteria(Insurance.class)

The above statement returns a reference to Criteria for the Order ORM class.

Once created, you add **Criterion** objects (generally obtained from static methods of the **Expression** class) to build the query. Methods such as **setFirstResult()**, **setMaxResults()**, and **setCacheable()** may be used to customize the query behavior in the same way as in the **Query** interface. Finally, to execute the query, the **list()** method is invoked.

For example:

```
crit.add(Expression.gt("investementAmount",
new Integer(900)));
    List list = crit.list();
```

The above statements are fired on the **investementAmount** field of the Insurance class, which specify the list of those records that have the investment amount greater than 900.

Expressions:

The Hibernate Query API supports a rich set of comparison operators with the **Expression** class. The standard SQL operators (=, <, d", >, e") are supported by the following methods in the Expression class, respectively: **eq(), lt(), le(), gt(), ge()**.

Following important methods of the Expression class are shown in the table:

Method	**Description**
Expression**.between**	This is used to apply a "between" constraint to the named property
Expression**.eq**	This is used to apply an "equal" constraint to the named property
Expression**.ge**	This is used to apply a "greater than or equal" constraint to the named property
Expression**.gt**	This is used to apply a "greater than" constraint to the named property
Expression**.lt**	This is used to apply a "less than" constraint to the named property
Expression**.in**	This is used to apply an "in" constraint to the named property
Expression**.le**	This is used to apply a "less than or equal" constraint to the named property
Expression**.and**	This returns the conjunctions of two expressions.
Expression**.or**	This returns the disjunction of the two expressions.

2. Criterion

Another core API, in the relational approach conditions is know as **criterion**. A criterion is an instance of the interface **org.hibernate.criterion.Criterion**. To retrieve data based on certain conditions, Restriction must be used on the criteria query. In other words, Criterion is the object-oriented representation of the "where" clause of a SQL query.

3. Restrictions:

The class **org.hibernate.criterion.Restrictions** defines factory methods for obtaining certain built-in

Criterion types.

In code this would be:**Criterion crit=Restriction.eq(**lngInsuranceId**,5); criteria.add(crit);**

Following important methods of the Expression class are shown in the table:

Method	Description
Restriction.**allEq**	This is used to apply an "equals" constraint to each property in the key set of a Map
Restriction.**between**	This is used to apply a "between" constraint to the named property
Restriction.**eq**	This is used to apply an "equal" constraint to the named property
Restriction.**ge**	This is used to apply a "greater than or equal" constraint to the named property
Restriction.**gt**	This is used to apply a "greater than" constraint to the named property
Restriction.**idEq**	This is used to apply an "equal" constraint to the identifier property
Restriction.**ilike**	This is case-insensitive "like", similar to Postgres ilike operator
Restriction.**in**	This is used to apply an "in" constraint to the named property
Restriction.**isNotNull**	This is used to apply an "is not null" constraint to the named property
Restriction.**isNull**	This is used to apply an "is null" constraint to the named property
Restriction.**le**	This is used to apply a "less than or equal" constraint to the named property
Restriction.**like**	This is used to apply a "like" constraint to the named property
Restriction.**lt**	This is used to apply a "less than" constraint to the named property
Restriction.**ltProperty**	This is used to apply a "less than" constraint to two properties
Restriction.**ne**	This is used to apply a "not equal" constraint to the named property
Restriction.**neProperty**	This is used to apply a "not equal" constraint to two properties
Restriction.**not**	This returns the negation of an expression
Restriction.**or**	This returns the disjuction of two expressions

The Hibernate criteria query API provides two query types that allow programmatic fetching of persistent objects: **query by criteria (QBC)**

and **query by example (QBE)**.

Lets take an example of **query by criteria (QBC)** using **Expressions**. This example is taking the same table **Insurance** as we have used in our previous issue.

Consider the following table Insurance having such records:

ID	insurance_name	invested_amount	investement_date
2	Life Insurance	250000000-00-00	00:00:00
1	Jivan Dhara	200002007-07-30	17:29:05
3	Life Insurance	500 2005-10-15	00:00:00
4	Car Insurance	2500 2005-01-01	00:00:00
5	Dental Insurance	500 2004-01-01	00:00:00
6	Life Insurance	900 2003-01-01	00:00:00
7	Travel Insurance	2000 2005-02-02	00:00:00

Here is the code of the class using "eq" Expression:

```
package javajzzup.hibernate;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Expression;

public class
HibernateCriteriaQueryExpressionEq {

 /**
  * @param args
  */
 public static void main(String[] args) {
   // TODO Auto-generated method stub

   Session sess = null;
   try{
     SessionFactory fact = new
Configuration().configure().buildSessionFactory();
     sess = fact.openSession();
     Criteria crit =
sess.createCriteria(Insurance.class);
     DateFormat format = new
SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
      Date date = (Date)format.parse("2005-
01-01 00:00:00");

crit.add(Expression.eq("investementDate",date));
     List list = crit.list();
     for(Iterator it =
list.iterator();it.hasNext();){
       Insurance ins = (Insurance)it.next();
       System.out.println("Id: " +
ins.getLngInsuranceId());
       System.out.println("Insurance Name: "
+ ins.getInsuranceName());
       System.out.println("Insurance Amount:
" + ins.getInvestementAmount());
  System.out.println("Investement Date: " +
ins.getInvestementDate());
     }
     sess.clear();
   }
   catch(Exception e){
     System.out.println(e.getMessage());
   }
 }
}
```

In the above code the **crit.add(Expression.eq ("investementDate",date));** uses two parameter e.g. eq("property_name",Object val). It fetches those records from the table that meets the specified date.

The output of this program will be shown as:

```
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.Environment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: select this_.ID as ID0_0_, this_.insurance_name as insurance2_0_0_, this_
InsuranceId: 4
InsuranceName: Car Insurance
InsuranceAmount: 2500
InvestementDate: 2005-01-01 00:00:00.0
```

Download Program

4. Querying with Projections:

Projections are used to customize the results from the database. In general Projection means to retrieve; while in case of **SQL** Projection means "**Select**" clause. The above code retrieves all the rows from the 'insurance' table. But what if only the data contained in one of the fields has to be retrieved, as in the following SQL query:

SELECT NAME FROM PRODUCT

Here, the Projection class comes into play. The above query can be rewritten into a Criteria query as:

```
ProjectionList proList =
Projections.projectionList();
proList.add(Projections.property("name"));
crit.setProjection(proList);
```

In the above code, **ProjectionList** is the list of projection instances, which are result of Query's object. The fieldname is passed as an argument to the **property()** method of the **Projection** class. The Projection instance returned in turn becomes an argument to the **setProjection()** method.

Now lets see an example of hibernate projection:

In the class **projectionExample.java,** first a session object created with the help of the **SessionFactory** interface. Then use the **createQuery()** method of the Session object which returns a Query object. Now we use the openSession() method of the SessionFactory interface simply to instantiate the Session object.

Then the criteria object is obtained simply by invoking the **createCriteria()** method of the Session's object. Now create a **projectionList** object add the fields having properties **"name"** and **"price".** Set it to the Criteria object by invoking the **setProjection()** method and passing the projectList object into this method and then add this object into the List interface's list object and iterate this object list object to display the data contained in this object.

Here is the full source code of **projectionExample.java**:

```
package javajzzup.hibernate;

import java.util.Iterator;
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.ProjectionList;
import org.hibernate.criterion.Projections;

public class projectionExample {

 /**
  * @param args
  */
 public static void main(String[] args) {
   // TODO Auto-generated method stub

   Session sess = null;
   try{
     SessionFactory sfact = new
Configuration().configure().buildSessionFactory();
      sess = sfact.openSession();
     Criteria crit =
sess.createCriteria(Product.class);
      ProjectionList proList =
Projections.projectionList();
    proList.add(Projections.property("name"));
     proList.add(Projections.property("price"));
      crit.setProjection(proList);
      List list = crit.list();
      Iterator it = list.iterator();
      if(!it.hasNext()){
        System.out.println("No any data!");
```

```
    }
    else{
      while(it.hasNext()){
        Object[] row = (Object[])it.next();
        for(int i = 0; i < row.length;i++){
          System.out.print(row[i]);
          System.out.println();
        }
      }
    }
    sess.close();
  }
  catch(Exception e){
    System.out.println(e.getMessage());
  }
 }
}
```

Here is the code for **Product.java** to set the class's objects with the table's fields.

package javajzzup.hibernate;

```
public class Product {

 private int id;
 private String name;
 private double price;
 private Dealer dealer;
 private int did;

 public Product(String name, double price)
{
   super();
    // TODO Auto-generated constructor stub
  this.name = name;
  this.price = price;
 }
 public Product() {
  super();
   // TODO Auto-generated constructor stub
 }
 public Dealer getDealer() {
  return dealer;
 }
 public void setDealer(Dealer dealer) {
  this.dealer = dealer;
 }

 public double getDid() {
  return did;
 }
 public void setDid(int did) {
  this.did = did;
 }

 public int getId() {
  return id;
 }
 public void setId(int id) {
  this.id = id;
 }
 public String getName() {
  return name;
 }
 public void setName(String name) {
  this.name = name;
 }
 public double getPrice() {
  return price;
 }
 public void setPrice(double price) {
  this.price = price;
 }

}
```

The output of this program will be shown as:

```
log4j:WARN No appenders could be found for
logger (org.hibernate.cfg.Environment).
log4j:WARN Please initialize the log4j system
properly.
Hibernate: select this_.name as y0_,
this_.price as y1_ from Product this_
Product Name      Price
Computer          23000.0
Mobile            15000.0
Laptop            200.0
Keyboard          1500.0
PenDrive          200.0
HardDisk          2500.0
```

Download example

5. Order

The Order class defined in the **org.hibernate.criterion** package represents the "order by" clause of SQL. By using the **asc()** and **desc()** methods of this class, order can be imposed upon the Criteria resultselt.

Hibernate Query Language

To order your query results, you use the addOrder() method and the Order class:

```
List orde r=
session.createCriteria(Product.class)
        .add(Expression.lt("price", new
Integer(2000)))
        .addOrder( Order.asc("name") )
        .list();
```

The generated HQL query would be something like:

from Product p order by p.name asc

URL Example with Desktop class

1. Using the Desktop class to launch a URL with default browser in Java

This article describes the new **Desktop** API, which allows Java applications to interact with the default applications associated with specific file types on the host platform.
This new functionality is provided by the java.awt.Desktop class, which is adopted from the **JDesktop Integration Components (JDIC)** project.

The new Desktop API allows your Java applications to do the following:

- Launch the host system's default browser with a specific Uniform Resource Identifier (URI).
- Launch the host system's default email client
- Launch applications to open, edit, or print files associated with those applications

The sample code given below demonstrates how to launch the host system's default browser with a specific Uniform Resource Identifier (URI). Whenever this program is run, it will automatically open a specified URL in the system's default browser.

Lets see the code of **DesktopClassToLaunch.java** file.

```
import java.awt.Desktop;
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
public class DesktopClassToLaunch {
  public static void main(String[] a) {
    try {
     URI uri = new URI("http://
www.javajazzup.com");
      Desktop desktop = null;
      if (Desktop.isDesktopSupported()) {
        desktop = Desktop.getDesktop();
      }
      if (desktop != null)
        desktop.browse(uri);
    } catch (IOException ioe) {
      ioe.printStackTrace();
    } catch (URISyntaxException use) {
      use.printStackTrace();
    }
  }
}
```

Desktop.isDesktopSupported() method to determine whether the Desktop API is available. After determining it, the application can retrieve a Desktop instance using the static method **getDesktop().**

After compiling and running this program, the browser will be open with the URL javajazzup.com automatically like:

Download Example

2. Display ToolTip within a specified area on JFrame.

In Java, **javax.swing** package provides a class known as **JToolTip**. This swing's component is used to display a "Tip" for another Component and provides API to computerize the process of using ToolTips.

For example, the **JToolTip.setToolTipText** method is used to specify the text for a standard tooltip. To create a custom ToolTip, the JComponent's **createToolTip** method is overridden. It returns the instance of JToolTip that should be used to display the tooltip.

URL Example with Desktop class

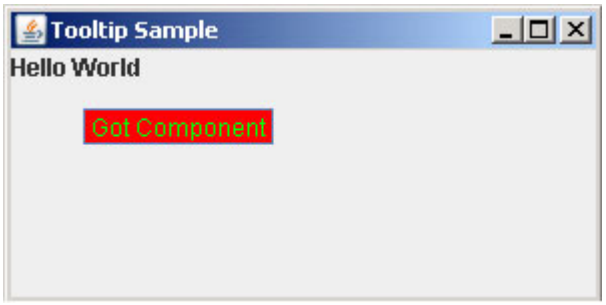
The given code in the **DispalyTooltip.java** file has a label component. Whenever we move the mouse on the particular component, a related tooltip is displayed with the specified text.

DispalyTooltip.java

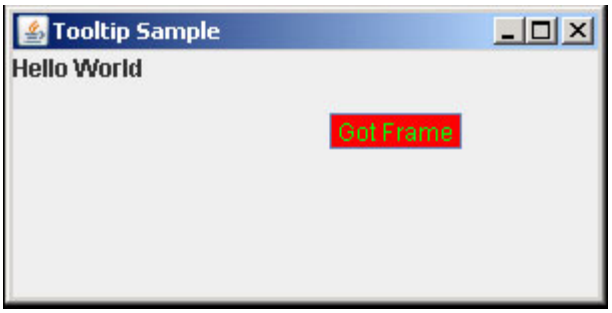
```
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.*;
import javax.swing.JLabel;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JToolTip;
public class DisplayTooltip {
  public static void main(String args[]) {
    String title = "Tooltip Sample";
   JFrame frame = new JFrame(title);
    frame.setDefaultCloseOperation
(JFrame.EXIT_ON_CLOSE);
    Container container =
frame.getContentPane();
   JPanel panel = new JPanel();
   panel.setToolTipText("<HtMl>
Tooltip<br>Message");
   container.add(panel,
BorderLayout.CENTER);
   JLabel label = new JLabel("Hello World") {
     public JToolTip createToolTip() {
       JToolTip tip = super.createToolTip();
        tip.setBackground(Color.red);
        tip.setForeground(Color.green);
       return tip;
     }
     public boolean contains(int x, int y) {
       if (x < 100) {
         setToolTipText("Got Component");
       } else {
         setToolTipText("Got Frame");
       }
       return super.contains(x, y);
     }
   };
   label.setToolTipText("Hello World");
   frame.getContentPane().add(label,
BorderLayout.NORTH);
    frame.setSize(300, 150);
   frame.setVisible(true);
  }
}
```

Lets see the outputs from the three kinds of ToolTips.

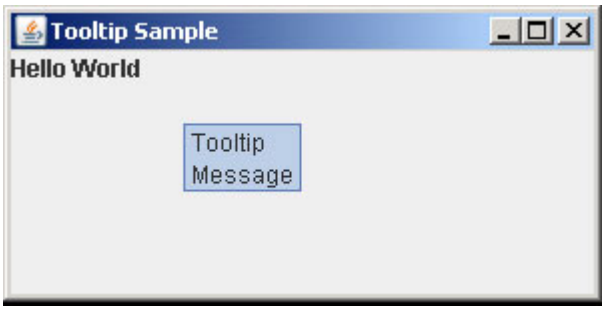
1. Move the mouse on the Label component.



2. Move mouse outside of the component area.



3. Move mouse outside of the frame area.



Download Example

3. Shuffle the elements of a Collection using ArrayList interface.

URL Example with Desktop class

The **Collections** class which can be found within the **java.util** namespace provides two methods that **suffle** the elements of a Collection.

```
static void shuffle(List<?> list)
static void shuffle(List<?> list, Random
rnd)
```

The first method shuffles the elements according to a default source of randomness, while the second uses a specified source of randomness. The random number methods generate numbers with replacement. This means that, a particular random number may be generated repeatedly.

The example below shows how to produce the values from 0 to 50 in a random order.

ShuffleExample.java

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Random;
public class ShuffleExample {
  public static void main(String args[]) {
    String str[] = { "A", "B", "C", "D", "E" };
// Create a list1 with elements
    List list1 = Arrays.asList(str);
      Random rand = new Random(50);
  // Shuffle the elements in the list
    Collections.shuffle(list1, rand);
   System.out.println(list1);
        Collections.shuffle(list1, rand);
      System.out.println(list1);
    }
}
```

In this program, the output will be generated different each times when we shuffle the list of a Collection.

```
C:\Tips&Tricks>javac ShuffleTest.java
C:\Tips&Tricks>java ShuffleTest
[A, B, D, E, C]
[A, D, E, C, B]
```

Download Example

4. Password Prompting with java.io.Console

The new released JDK6 includes a new **Console** class, which can be found in the **java.io** package. This class adds some new features to enhance and simplify command-line applications. It includes a method specifically for reading passwords that disables console **echo** and returns a char array for security purpose.

The example given below read the password from the console but the password will not be echoed to the console screen. If the given password matches with the specified characters as **"javajazzup"** then a message **"Access granted"** is displayed; otherwise it displays **"Access denied"** as an output.

PasswordPrompting.java

```
import java.io.Console;
import java.util.Arrays;
public class PasswordPrompting {
  public static void main(String[] args) {
    Console console = System.console();
   if (console == null) {
     System.out.println("Console is not
available");
     System.exit(1);
   }char[] password =
"javajazzup".toCharArray();
  /* Read password, the password will not be
echoed to the console screen and returned as
an array of characters.*/
   char[] passwordEntered =
console.readPassword("Enter password: ");
   if (Arrays.equals(password,
passwordEntered)) {
     System.out.println("\n Access granted
\n");
    // Clear the password after validation
successful
     Arrays.fill(password, ' ');
     Arrays.fill(passwordEntered, ' ');
   } else {
     System.out.println("Access denied");
     System.exit(1);
   }
  }
}
```

URL Example with Desktop class

See the output when you type "javajazzup" as a password:

```
C:\Tips&Tricks>java PasswordPromptingDemo
Enter password:
```

Access granted

And when you type another password:

```
C:\Tips&Tricks>java PasswordPromptingDemo
Enter password:
```

Access denied

Download Example

5. Drag and drop between JTextArea and JtextField

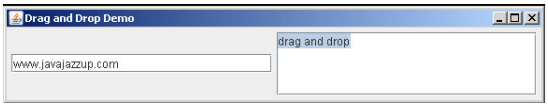
In Java, DnD is a data transfer API. The user transfers the data from one place to another by using the mouse (usually). One selects something with a mouse press, drags it (moves the mouse while keeping the mouse button pressed) and releases the mouse button someplace else. When the button is released the data is "dropped" at that location.

The example given below shows how to use the drag-and-drop support built into Swing components. This program contains a TextArea and TextField with some specified text. The **setDragEnabled** method having a **true** argument of each swing's make it as a drag and droppable component.

DnDFieldDemo.java

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
public class DnDFieldDemo {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Drag and
Drop Demo");
    frame.setDefaultCloseOperation
(JFrame.EXIT_ON_CLOSE);
   frame.setContentPane(new JPanel());
   JTextField textField = new JTextField(25);
   textField.setText("www.javajazzup.com");
   frame.add(textField);
    JTextArea textArea = new JTextArea(4,
25);
    textArea.setText("drag and drop");
   frame.getContentPane().add(new
JScrollPane(textArea));
    textArea.setDragEnabled(true);
   textField.setDragEnabled(true);
   frame.pack();
   frame.setVisible(true);
  }
}
```

The output will be show as:



Now you can drag and drop the text between TextField and TextArea.

Download Example

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

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 technocrates 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 technocrates 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 technocrates 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

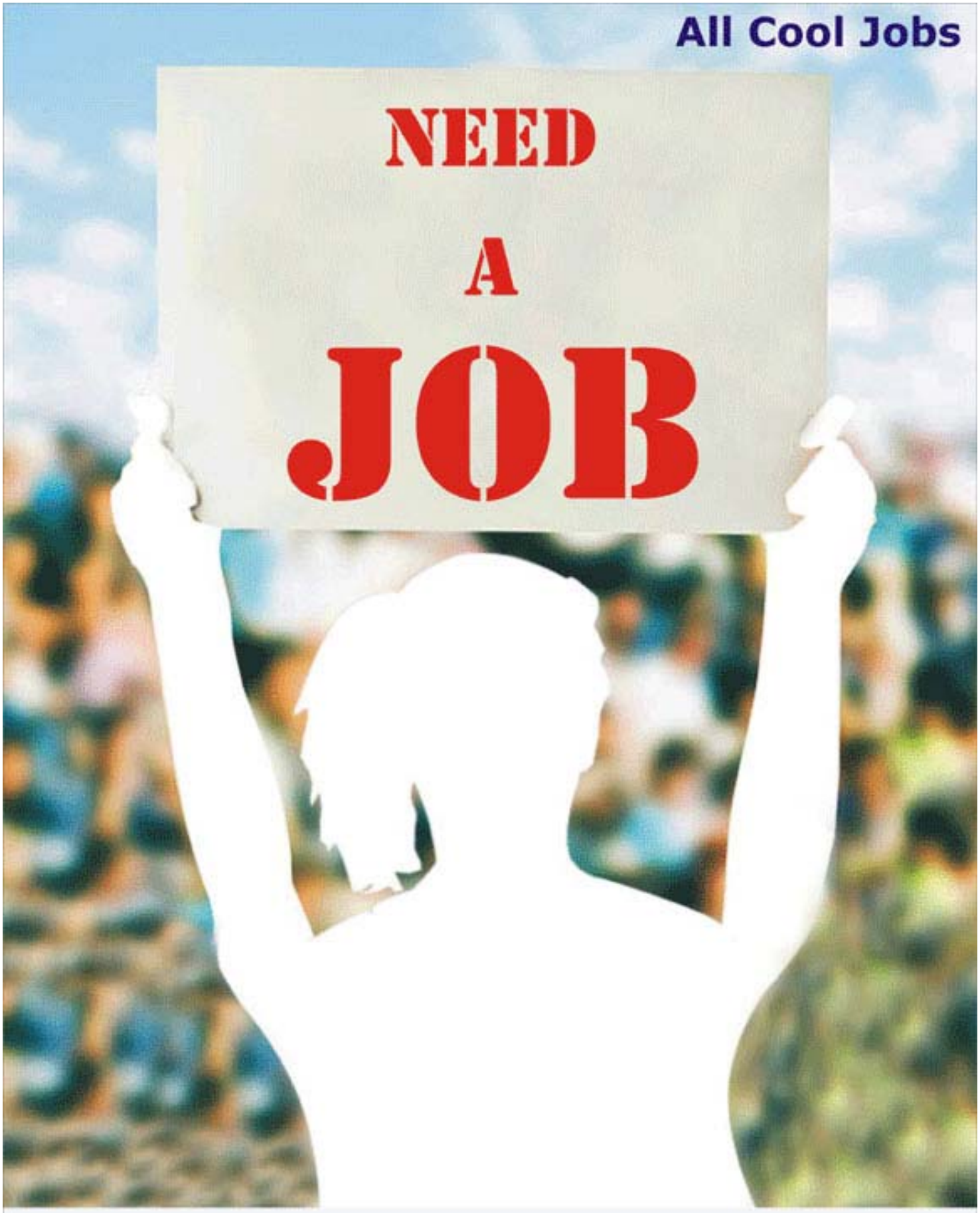
Were inviting people from lots of different worlds. We do not expect everybody at Readers Forum to be an expert in some areas. Your expertise is a real resource you may contribute to the Java Jazzup. We want your curiosity to be a resource, too. You can also present a question, problem, or puzzle that revolves around java technologies along with their solution that you think would get really appreciated by the java readers around the globe.

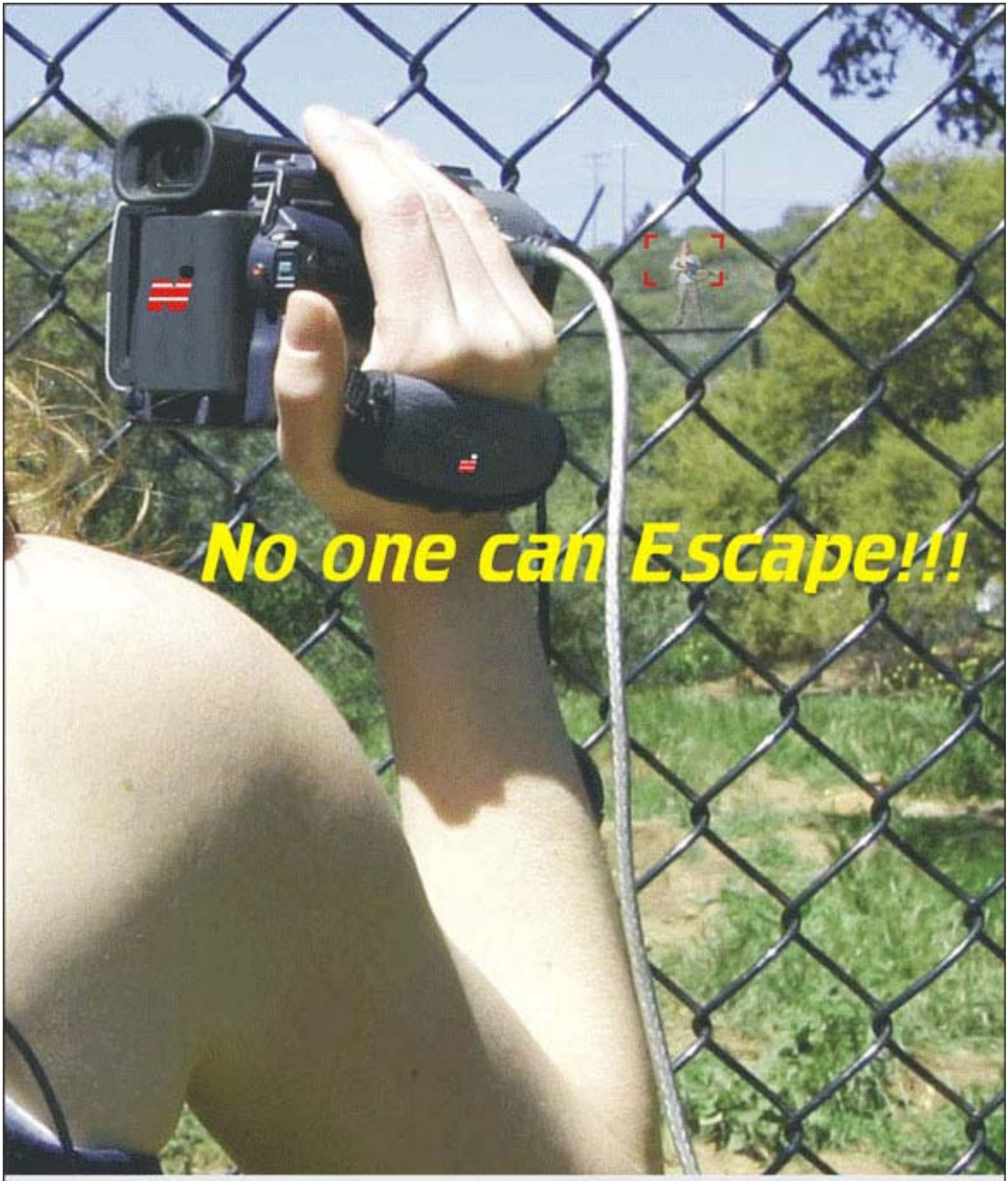
Post resourceful URLs

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.


All Cool Jobs
NEED
A
JOB
log on to http://www.allcooljobs.com/



NEWSTRACK India

Freedom to Express

http://www.newstrackindia.com