

# Java Jazz up

A BETTER WAY TO LEARN PROGRAMMING



AOP

Design Pattern

IoC

Maven

JavaME

EJB 3.0

JBoss seam

Ajax

Struts 2.0

JSF

Tips & Tricks

This Diwali  
Power your IT-Driven Process  
&  
Accelerate Business with  
A global leader in Services...

Web Development, Web Designing, Web Redesigning, Web Promotion, Search Engine Optimization, Search Engine Re-submission, Web Hosting, Linux Hosting, Windows Hosting, E-mail Hosting, Software Solutions, Web Hosting, JSF Development, Web Development, Outsourcing, E-Commerce Solutions, Customer Relationship Management, Content Development, Content Development, Technical Documentation.



**“ Innovative great ideas always  
encounter violent opposition from  
mediocre minds. ”**

**Published by**

**RoseIndia**

---

**JavaJazzUp Team**

---

**Editor-in-Chief**

Deepak Kumar

**Editor-Technical**

Ravi Kant  
Vinod Kumar

**Sr. Graphics Designer**

Suman Saurabh

**Graphics Designer**

Santosh Kumar

---

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

**“Free”**

## Editorial

Dear Readers,

Happy Deepawali

We are here again with the Nov' 07 issue of Java Jazz-up, celebrating the festival of light together. The current edition highlights the interesting Java technologies presented in form of articles developed by the Java Jazz-up developers' Team. 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.

The Editorial Choice section highlights the editor's viewpoint orienting around the java innovations in diverse spheres of human interest. In this issue, this section talks high of AOP and IoC technologies and tries to avail its role in the current java software development scenario.

The Java ME section highlights the role of java in the development of software for small, resource-constrained devices such as cell phones, PDAs and set-top boxes. This issue targets to provide a beginner's guide to start with 'Java Mobile Phones Gaming Applications' in a very easy and concise manner.

The set of articles conferring technologies like Maven2, Design patterns, JSF tags, Web Services, JBoss Seam, AJAX etc. are provided in such a manner that even a novice learns and implements the concepts in a very easy manner.

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** [AOP and IoC](#) | Aspect Oriented programming (AOP) decomposes a system into concerns (i.e. term referring to the core elements required to focus on), instead of objects. It deals with "aspects" that cross-cuts across the code which could be difficult or impossible to modularize with OOP.
- 10** [Java News](#) | Sentilla Corp. introduced a software suite for java applications to run on low-power microprocessors embedded on devices. Users can wirelessly manage those applications using the platform. Joe Polastre, chief technology officer and co-founder of Sentilla, said "It overcomes challenges on running Java in tiny devices with small memory by squeezing a full Java environment into microprocessors."
- 11** [New Releases](#) | Sun's released the Java Wireless Toolkit 2.5.2. The Toolkit includes build tools, utilities, and a device emulator. This WTK update provides improved support for multiuser environments and also linux binary that is supported on systems running glibc 2.3 libraries.
- 12** [Java Developers Desk](#) | Migrating to EJB 3.0 is a big step towards simplifying the process of developing EJBs, which reduces lots of complexities, time and cost. In spite of being rich featured, developers feel complex working with previous versions of EJB.
- 13** [Java ME](#) | A Mobile Gaming is the best medium of entertainment in the real life because mobile game is user friendly and portable. However, the mobile industry both in gaming and applications is still in its infancy.
- 16** [JBoss Seam: Stitching JSF and EJB3](#) | Open source, and developed under the auspices of JBoss, Seam is a component framework that focuses to deliver full-featured JEE 5 applications in a lightweight code base i.e. requiring only a fraction of the code of regular JEE applications.
- 21** [Maven Plug-in](#) | Plugins are great in simplifying the life of programmers; it actually reduces the repetitive tasks involved in the programming.
- 23** [Tomahawk Tags](#) | Tomahawk tags are the collection of standard components with extended functionality and supports all the existing JSF components with additional sets of functionality
- 29** [Struts2](#) | Apache Struts is an open-source framework used to develop Java web application. Originally developed by the programmer and author Craig R.
- 37** [Design Patterns](#) | Structural Patterns are design patterns, which describe the best possible ways to combine the objects and classes forming a larger complex structure in an easy manner
- 43** [JSF](#) | Many of the web-based applications consists of login module which lets the user enter with its own identity and also let the admin authenticate the user or differentiate between the registered and normal user.
- 47** [AJAX: Redefining Web Applications](#) | AJAX stands for Asynchronous JavaScript And XML. AJAX is not a new programming language, but a new way to use existing standards.
- 57** [Tips & Tricks](#) | Splash screens are standard part of many GUI applications to let the user know about starting of the application. AWT/Swing can be used to create splash screens in Java. Prior to Java SE 6, you need to create a window and include an image in it when main method starts to get the behavior of splash screen.
- 67** [Advertise with Us](#) | We are the top most providers of technology stuffs to the java community.



# Editor's Choice -AOP and IoC

## AOP : New Programming Design Paradigm

Aspect Oriented programming (AOP) decomposes a system into concerns (i.e. term referring to the core elements required to focus on), instead of objects. It deals with "aspects" that cross-cuts across the code which could be difficult or impossible to modularize with OOP.

### Separation of concerns (SoC)

AOP promotes separation of concerns within the systems where **separation of concerns (SoC)** is a process to break a computer program into distinct features that were previously overlapping in functionality, as little as possible. A concern is any piece of interest needed to focus on while developing a program. Typically, concerns are synonymous with features or behaviors, lets talk of a credit card processing system, the related *core concern* of the system deals with processing the monetary transactions, while its *application-level concerns* handles logging, transaction integrity, authentication, security, performance, and so on. Many such concerns are known as *crosscutting concerns*. Lets talk of Logging, it offers one example of a crosscutting concern. A logging strategy necessarily affects every single logged part of the system. Logging thereby crosscuts all the logged classes and methods.

Such crosscutting concerns affect the multiple implementation modules. With the current programming methodologies, crosscutting concerns span over multiple modules causing a system to be complex to design, understand, implement, evolve and even maintain with time.

Lets quickly give a glance to the OOPs concept. OOPs programming model creates programs around the real world entities. Here the programs are developed around the objects and data rather than actions and logics. In OOPs, every real life object has properties and behavior. This feature is achievable with different languages like C++, Java, C# etc. However in the current scenario, object-oriented programming (OOP) seems to be no more an extra-ordinary programming model where real world problems are decomposed into objects

encapsulating behavior and data in a single unit. Although the developers have met great success in the past in modeling and implementing complex software systems, However they faces crucial problems in maintaining the code while working with large projects. Most of the time, to make a minor change to a program may require maintaining several updates to a large number of unrelated modules. Now with the advent of AOP, such problems are easily rectified, as it allows the developers to solve the complex problems involved with software development that couldn't be resolved easily with the object-oriented programming techniques.

**However, AOP** being a new programming technique, allows programmers to modularize crosscutting concerns. It allows the programmers to dynamically modify the static OO model to create a system that can grow to meet new requirements. It allows separating the crosscutting concerns into the single units called **aspects**.

It is a modular unit of crosscutting implementation, which encapsulates behaviors that affect multiple classes into reusable modules. AOP is not bound to a specific programming language. It is a concept that can be implemented with different languages (for example C++, Smalltalk, Java etc.).

Aspect-Oriented Software Development (AOSD) attempts to aid programmers in the separation of concerns, specifically cross-cutting concerns, as an advance to the modularization. AOP does so using primarily language changes, while AOSD uses a combination of language, environment, and method.

The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules. Systems are composed of several components each responsible for a specific piece of functionality. Irrespective of the core functionality of a program, the system services like logging, transaction management, security etc., must be included in the program. These system services are commonly referred to as 'cross-cutting concerns' as they tend to

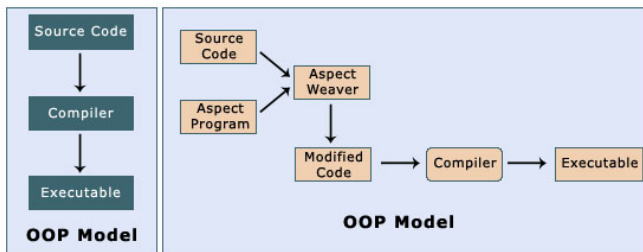
# AOP and IoC

cut across multiple components in a system.

AOSD makes it possible to modularize and separate these services and then apply them declaratively to the components and we can focus on our own specific concerns. For instance if we talk of spring framework, there aspects are wired into objects in the spring XML file in the same way as JavaBean does. This process is also known as 'Weaving'.

In a typical object-oriented development approach the developer might implement logging functionality by putting logger statements in all your methods and Java classes. In an AOP approach you would instead modularize the logging services and apply them declaratively to the components that required logging. The advantage, of course, is that the Java class doesn't need to know about the existence of the logging service or concern itself with any related code. As a result, application code written with Spring AOP is loosely coupled.

## Lets do a comparison of OOP's model with an AOP model, diagrammatically:



Here, the OOPs model shows that a source code is directly compiled to an executable form. On the other hand, the AOP model depicts the flow such that the source code in the OOP model gets compiled with the aspects such that it modifies the behavior of the OOPs model and modularizes the crosscutting concerns.

Among the most OOPs languages, **Java** is a true Object-Oriented Programming language with a best support for AOP techniques. There are multiple tools available to support AOP with Java, few of them are shown below:

- **AspectJ**
- **AspectWerkz**
- **Hyper/J**
- **JAC**
- **JMangler**
- **MixJuice**
- **PROSE**
- **ArchJava**

## II. IoC: Inversion Of Control

"The IoC pattern enables better software design that facilitates reuse, loose coupling, and easy testing of software components."

The basic concept of the Inversion of Control pattern (also known as dependency injection) is that you do not create your objects but describe how they should be created. You avoid connecting components and services together in your code. Instead the configuration files are used to describe the services needed by a component .

The Inversion of Control (IoC) pattern, also known as Dependency Injection, has recently become popular in the JEE community. Several open source projects, including Spring, PicoContainer, and HiveMind, use the IoC pattern to develop lightweight J2EE Containers. IoC is not a new concept, however. It has been around for several years now. Using object-oriented design principles and features such as interface, inheritance, and polymorphism, the IoC pattern enables better software design that facilitates reuse, loose coupling, and easy testing of software components.

Inversion of Control (IoC) means that objects do not create other objects on which they rely to do their work. Instead, they get the objects that they need from an outside source (for example, an xml configuration file).

Dependency Injection (DI) means that this is done without the object intervention, usually by a framework component that passes constructor parameters and set properties.

# AOP and IoC

## IoC: salient features

1. Eliminates lookup code from within your application.
2. Allows pluggability and hot swapping.
3. Promotes good OO design.
4. Enables the reuse of existing code.
5. Makes an application extremely testable.

In a typical IOC scenario, the container creates all the objects, wires them together by setting the necessary properties. It also determines when methods will be invoked. The three-implementation pattern types for IOC are listed in the table below.

**Type 1:** Services need to implement a dedicated interface through which they are provided with an object from which they can look up dependencies (for example, additional needed services).

**Type 2:** Dependencies are assigned through JavaBeans properties (for example, setter methods).

**Type 3:** Dependencies are provided as constructor parameters and are not exposed as JavaBeans properties.

For Instance, the Spring framework uses the Type 2 and Type 3 implementations for its IoC container. IoC is a broad concept, its two main types are:

**1. Dependency Lookup:** In the Type 1 IoC The container provides callbacks to components and a lookup context. The managed objects are responsible for their other lookups. This is the EJB Approach. The Inversion of Control is limited to the Container involved callback methods that the code can use to obtain resources. Here JNDI is used to look up other EJBs and resources. Because of this reason EJB is not branded as 'IOC framework'. There are some problems in this implementation. The class needs a application server environment as it is dependent on JNDI and it is hard to test as we need to provide a dummy JNDI contest for testing purpose.

**2. Dependency Injection:** Type 2 / Type 3 IoC In this application objects are not responsible to looking up resources they depend on. Instead IoC container configures the object externalizing resource lookup from application code into the container. That is, dependencies are injected into objects. Thus lookups are completely removed from application objects and it can be used outside the container also. Here, the objects can be populated via Setter Injection (Java-Beans properties) or Constructor Injection (constructor arguments). Each method has its own advantage and disadvantage.

Here, the objects can be populated via Setter Injection (Java-Beans properties) or Constructor Injection (constructor arguments). Each method has its own advantage and disadvantage.

**Setter Injection:** Normally in all the java beans, we use setter and getter method to set and get the value of property as follows:

```
public class nameBean {
    String name;
    public void setName(String a) {
        name = a;
    }
    public String getName() {
        return name;
    }
}
```

We create an instance of the bean 'nameBean' (say bean1) and set property as

```
bean1.setName("amit");
```

Here in setter injection, we set the property 'name' by using the <property> subelement of <bean> tag in spring configuration file as shown below:

```
<bean id="bean1" class="nameBean">
    <property name="name" >
        <value>amit</value>
    </property>
</bean>
```

# AOP and IoC

The subelement <value> sets the 'name' property by calling the set method as setName("amit"); This process is called **setter injection**.

**constructor injection** : For constructor injection, we use constructor with parameters as shown below:

```
public class nameBean {
String name;
public nameBean (String a) {
name = a;
}
}
```

We will set the property 'name' while creating an instance of the bean 'nameBean' as

```
nameBean bean1 = new nameBean("amit");
```

Here we use the <constructor-arg> element to set the the property by constructor injection as:

```
<bean id="bean1" class="nameBean ">
<constructor-arg>
<value>Bean Value</value>
</constructor-arg>
</bean>
```

To set properties that reference other beans <ref>, subelement of <property> is used as shown below:

```
<bean id="bean1" class="bean11">
<property name="game">
<ref bean="bean2"/>
</property>
</bean>
<bean id="bean2" class="bean22" />
```

## The IOC containers

The mainstream JEE involves heavyweight containers to develop applications. So exploring alternatives and coming up with creative ideas have evolved a lot of open source java communities. In the Java community there's been a rush of lightweight containers

that help to assemble components from different projects into a cohesive application. Several open source projects, including Spring, PicoContainer, and HiveMind use the IoC pattern to develop lightweight JEE Containers. The container manages the life cycle and configuration of application objects.

Let's see how Spring container implements the IoC concepts

Spring should not, however, be confused with traditional heavyweight EJB containers, which are often large. The Spring actually comes with two distinct containers:

**1. Bean Factories** - defined by "**org.springframework.beans.factory.BeanFactory**" are the simplest containers, providing support for dependency injection.

**2. Application contexts** - defined by "**org.springframework.context.ApplicationContext**" provides the application framework services.

## Configuration metadata

As can be seen in the above image, the Spring IoC container consumes some form of configuration metadata; this configuration metadata is nothing more than how you (as an application developer) inform the Spring container as to how to "instantiate, configure, and assemble [the objects in your application]". This configuration metadata is typically supplied in a simple and intuitive XML format. When using XML-based configuration metadata, you write bean definitions for those beans that you want the Spring IoC container to manage, and then let the container do it's stuff.

## BEAN FACTORY:

Bean factory is an implementation of the factory design pattern. Its function is to create and dispense beans. As the bean factory knows about many objects within an application, it is able to create association between collaborating objects, as they are instantiated. This moves the burden of configuration from the bean and



## AOP and IoC

the client.

**BEAN FACTORY** supports two object modes.

**Singleton mode** provides a shared instance of the object with a particular name, which can be retrieved on lookup. Singleton is the default and most often used object mode. It is ideal for stateless service objects.

**Prototype** mode ensures that each retrieval will result in the creation of an independent object. Prototype mode would be best used in a case where each user needed to have his or her own object. The bean factory concept is the foundation of Spring as an IOC container. IOC moves the responsibility for making things happen into the framework and away from application code. The Spring framework uses JavaBean properties and configuration data to figure out which dependencies must be set.

There are several implementation of BeanFactory like "**org.springframework.beans.factory.xml.XmlBeanFactory**" which loads its beans based on the definition contained in an XML file.

### APPLICATION CONTEXT:

The Application Context is spring's more advanced container. Like '**BeanFactory**' it can load bean definitions, wire beans together and dispense beans upon request. Additionally, It also provides:

1. A means for resolving text messages, including support for internationalization ie. i18n messages
2. A generic way to load file resources.
3. Events to notify beans that are registered as listeners.

Because of additional functionality, 'Application Context' is preferred over a BeanFactory. BeanFactory is used for simple applications and when the resource is scarce like mobile devices.


### III Service Abstraction Layers

Spring provides consistent integration with various standard and 3rd party APIs through its various Service abstraction layers. Few of them are:


1. Transaction Management abstraction for JTA, JDBC, others
2. Data Access abstraction for JDBC, Hibernate, JDO, TopLink, iBatis
3. Abstraction for Emailing
4. Remoting abstraction layers for EJB, Web Services, RMI, Hessian/Burlap

Benefits of the Service Abstraction Layers in spring framework:

1. There is no implicit contract with JNDI, etc.
2. It separates the user from the underlying APIs.
3. It enhances the reusability to a great extent.
4. Spring abstractions always consist of interfaces.
5. Testing is kept simpler than ever before.
6. For data access, Spring uses a generic transaction infrastructure and DAO exception hierarchy that is common across all supported platforms.



*We are the best in  
Shaping the  
Java Technology Stuff*



<http://www.roseindia.net>

# JAVA AROUND THE GLOBE

## **Sentilla Puts Java on Chips**

Sentilla Corp. introduced a software suite for java applications to run on low-power microprocessors embedded on devices. Users can wirelessly manage those applications using the platform. Joe Polastre, chief technology officer and co-founder of Sentilla, said "It overcomes challenges on running Java in tiny devices with small memory by squeezing a full Java environment into microprocessors. The platform uses memory management and storage on a device to swap Java code in and out of memory as needed. That allows the platform to use large applications without draining resources". Currently it works with Texas Instruments Inc.'s MSP430 microprocessor, a 16-bit RISC processor but in future more microprocessors will support it.

## **Apache Geronimo Passes J2EE Test Kit**

The Apache Software Foundation's Geronimo 1.0-M5, open source J2EE application server, has passed Sun Microsystem's the J2EE 1.4.1 test compatibility kit, or TCK. It was announced during a "birds of a feather" presentation at the recent JavaOne show in San Francisco. Geronimo provides support for Web, EJB, JMS and EIS applications, combined with enterprise grade configuration and management. "People shouldn't think of Geronimo as just another J2EE app server, but as the start of a system framework that can be used to build a variety of tailored infrastructure services," Gluecode's former CTO, Jeremy Boynes

## **BluePhoenix Wins Major Contract to Modernize PowerBuilder Applications to Java for Global Telecommunications Customer**

BluePhoenix Solutions announced that it has won a major contract to convert PowerBuilder applications to Java for a global telecommunications company. In this, BluePhoenix's LanguageMigrator conversion solution will be implemented, which enables organizations to automatically migrate legacy applications to prevailing languages and technologies. PowerBuilder is a 4GL rapid application development environment developed in early 90s. It has been used by thousands of companies but still many companies want to migrate their PowerBuilder applications to Java.

This is because of growing maintenance cost, functionality limitations, and fewer PowerBuilder programmers.

"Our customer's changing business requirements dictates a move to a modern platform and our automatic migration solution supports their strategic direction to consolidate on Java," said Arik Kilman, CEO of BluePhoenix. "Our LanguageMigrator solution along with our proven migration methodology enables us to guarantee results while minimizing risk and disruption to current operations."

## **Sun starts bidding adieu to mobile-specific Java**

Java Standard Edition (SE) for desktop computers will gradually replace Java Micro Edition (ME) as technology improvements let more computing power be packed into smaller devices, said James Gosling, the Sun vice president and known as the title "father of Java".

"We're trying to converge everything to the Java SE specification. Cell phones and TV set-top boxes are growing up," Gosling said at a Java media event here Wednesday. "That convergence is going to take years."

# New Releases

## **Sun's released the Java Wireless Toolkit 2.5.2.**

Sun has released the Java Wireless Toolkit 2.5.2. The Toolkit includes build tools, utilities, and a device emulator. This WTK update provides improved support for multiuser environments and also linux binary that is supported on systems running glibc 2.3 libraries. This release contains all advanced development features of its previous versions 2.2, 2.5, 2.5.1 such as MIDlet signing, certificate management, integrated over-the-air (OTA) emulation, push registry emulation, and more. It also includes Nokia's scalable network application package (SNAP) mobile API and the SNAP Mobile Sample Application as part of its external API feature.

## **Nuxeo releases version 5.1 of its open source ECM platform**

Nuxeo, the leader of open source ECM (Enterprise Content Management) provided complete ECM solutions for many large companies, has announced Nuxeo Enterprise Platform 5.1, a new version of its ECM platform. It includes extensible service-oriented architecture (SOA) and enhanced performance and functionality like advanced search service, Data import/export service, Enhanced horizontal scalability, Electronic and physical records management. Delivering enterprise-grade functional & technical support, certified software patches and updates and management tools are also available during every stage of the application lifecycle. It is completely extensible because it is built on infrastructure of plug-ins and extension points based on the OSGi standard.

"Nuxeo Service Platform, built on industry standards (such as Java EE 5 and OSGi), offers a complete range of component-based services making it possible to quickly build ECM applications. This SOA approach offers real and tangible benefits to our customers and our partners, software developers and integrators. Nuxeo strengthens its technological advantage, with the help of a dynamic ecosystem, and positions us to inject new energy and innovation into the ECM market," explains Eric Barroca, Executive VP in charge of Operations.

## **Sun releases NetBeans 6.0 Beta 2 under both GPL2 and CDDL.**

Sun and the NetBeans community released the latest build of NetBeans 6.0 Beta 2 and announced that this Java-based IDE is now dual-licensed with CDDL and the GNU General Public License (GPL) with Classpath exception.

## **JADE 6.2 launches with mobile compatibility, Java interoperability**

Jade Software has released the latest version of its software platform, JADE 6.2, which enables thin clients and standalone database applications to run on Windows Mobile devices. It also enables Java programs to interoperate with JADE applications. It provides a Java framework that encapsulates their Object Manager, Database and core programming model.

JADE program manager Dean Cooper says, "Jade is committed to interoperability. We have always had C and C++ language interoperability. Since JADE 6.0 we have invested in support for XML and web services. With JADE 6.2 we are enthusiastic about opening up to the Java community, both in industry and academia."

## **Apache Struts 2.0.11 GA release available**

The Apache Struts group has announced the release of Struts 2.0.11 and it is available as a "General Availability" release. This release includes a number of fixes and improvements since the 2.0.9 GA release. Apache Struts 2 is an elegant, extensible framework for creating enterprise Java web applications and is designed to streamline the full development cycle, from building, to deploying, to maintaining applications over time.

# Java Developers Desk - EJB 3.0

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

## Limitations of EJB 2.1:

Developing EJB before release of EJB 3.0 was not so easy because of some unnecessary steps involved that are usually unused. Some limitations of EJB 2.1 are listed below:

- 1 A set of three source files must be create
- 2 Creating multiple xml deployment descriptors
- 3 Implementing several callback methods
- 4 Throwing several types of exceptions
- 5 EJB-QL is limited in functionality and difficult to use

## Features of EJB 3.0:

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

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

## Migration from EJB 2.1 to EJB 3.0

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

1. In EJB 2.1, home interface extends the javax.ejb.EJBHome interface and local home interface extends the javax.ejb.EJBLocalHome interface. The

EJB 2.1 remote interface extends the javax.ejb.EJBObject interface and local interface extends the javax.ejb.EJBLocalObject interface. In EJB 3.0, home and component interfaces are replaced with POJI business interfaces.

2. EJB 2.1 needs the developer to implement a variety of callback methods in the bean class, like `ejbActivate()`, `ejbPassivate()`, `ejbLoad()`, and `ejbStore()`, most of which were never used. EJB 3.0 doesn't force to implement any of these methods and instead can designate any arbitrary method as a callback method to receive notifications for life cycle events.
3. In EJB 2.1, session bean implements the `SessionBean` interface and entity bean implements the `EntityBean` interface. In EJB 3.0, session and entity bean classes are POJOs and do not implement the `SessionBean` and `EntityBean` interfaces.
4. The deployment descriptor, which specifies the EJB name, the bean class name, the interfaces, the finder methods etc. is not required because they are replaced by metadata annotations in the bean classes. Annotations are available in JDK 5.0 so you need JDK 5.0 to develop EJB 3.0 EJBs.
5. In EJB 2.1, client application finds a reference to entity and session bean objects using JNDI name but in EJB 3.0, client finds them using dependency annotations like `@Resource`, `@Inject`, and `@EJB`.
6. In EJB 2.1, developers used their own way to perform database specific operations like primary key generation while EJB 3.0 provides support for several database-specific operations. The O/R mapping model has intrinsic support for native SQL. The O/R mapping is specified using annotations.
7. Runtime services like transaction and security are often implemented as the interceptor methods managed by the container. However, in EJB 3.0 developers can write custom interceptor. So developers have control for the actions like committing transaction, security check, etc.



# Java ME

## Mobile Gaming

A Mobile Gaming is the best medium of entertainment in the real life because mobile game is user friendly and portable. However, the mobile industry both in gaming and applications is still in its infancy. Developers, manufacturers and carriers are all still working hard to revolutionize the mobile industry and ultimately drive millions of Dollars in revenue. As a result, graphics are getting better and similarly game play is getting better.

## Always Start Simple

Always start with a simple standalone game that doesn't do much visually nor interactively and yet still gives you a good understanding how a game works and the other required information needed to start with a complete gaming application. It is better to finish a simple game and feel sense of accomplishment and then tackle a harder game.

## Game Categories

Well generally if you look closely at any game they all fall under a certain kind of game type. Some games may seem really cool to elaborate but if you break them down they are either re-used or developed with more ideas from the existing games. If you look at games like DukeNukem, Doom, Quake, Freelancer, Counter Strike, Return Castle Wolfenstein. They are really just the same ole games that were in 2D but now are in 3D.

Games can be roughly broken down into the categories like Arcade/Action Fast-paced, Rich graphics, highly interactive games.

For example, Card Games such as Poker, BlackJack Strategy Requires a lot the thinking and tactical moves and possible micro management. The above are more of the common categories of course there are others like trivia, Simulation.

## Mobile Game Constraints

Features such as memory, screen size, even colors act as the big hurdles while developing the nice java mobile games. However, all of these

are usually not a factor for console and PC application development. These constraints become more evident in a mobile game then in case of a mobile application as it requires a high interaction between user inputs, graphics, animation, sound and/or vibration. Additionally, while developing games you not only have to consider different manufacturers but also the different mobile handset models for the same manufacturer. Phone models can differ vastly from model to model in memory, color, screen size and user interface.

## I. Memory

### Types of Memory

In general working memory otherwise known as heap memory is the area of memory where the game stores information during execution of the game and is released when the game is terminated. You will have to refer to the manufacturers manual for the exact specifications. This is important to you because if the game is bigger then the allocated working memory on a device then the game simply won't run. Another memory you need to concern yourself with is the storage memory otherwise known as RMS the Record Management System. You need to be aware of the total allowable storage that is available for the particular handsets you wish to deploy to and possibly build an alternative logic into the game for cases when memory does run out.

## II. Display, Size and Color

Aside from memory another factor you must take into consideration is the size of the screen for each mobile handset. Take for example a Sony Ericsson P800 when folded out has pixel display of 208 × 320 and a Nokia 3650 has a display of 176 × 208. You may consider releasing specific version that includes the appropriate image sizes. The Nokia may have sprites the size of 16 × 16 pixels and the P800 may have sprites the size of 32 × 32. Though more and more mobile handsets are being released with color screens you should take in consideration the millions of existing phones already sold on the market that only have black and white displays.

# Java ME

## Coding Tips

The following tips are only suggestions and may or may not give gains in performance, it is to your own judgment and discretion to use them or not.

**1.** Use `StringBuffer` instead of `String` because of the fact the `String` object cannot be changed. Any modification to a `String` variable is actually a new object creation.

**2.** Accessing class variables directly is faster than using setter and getter methods.

**3.** Using local variables are more efficient than instance/class variables

**4.** Using variables are more efficient than arrays.

**5.** Avoid synchronization in loops because there is an extra overhead to lock and unlock each time the loop occurs

**6.** Counting down in loops is faster than counting up

**7.** Use compound operators like `x += 1` instead of `x = x + 1` because fewer byte codes are generated

**8.** Remove constant calculations in loops

**9.** Reuse objects

**10.** Assign null to unused objects, especially unused threads

**11.** Try to use already built-in methods, for example if you want to copy data from one array to another use `System.arraycopy` more than be more efficient than the one you created yourself

## Game applications with JAVA(TM)

Now, the gaming application for small devices is easy and fun with new MIDP 2.0 API. The biggest difference for a game developer between MIDP 1.0 and MIDP 2.0 lies with the enhanced graphics capabilities. With MIDP 1.0 you can certainly write some fun games—those of us.

Now with the 2.0 version, the Mobile Information Device Profile doesn't bring to the level of the latest game cube, but it brings the user's cell phone to the realm of Super Mario Brothers or there about.

A MIDlet is very much like an applet except that it runs on a mobile device rather than in a browser.

**The directory structure** of the gaming application consists of the following:

- **src**, containing source code
- **bin**, containing the manifest.mf file, JAD file, and JAR file
- **classes**, containing the compiled classes
- **res**, containing image, data, and other files required by the applicationJ2ME BASICS

## MIDlet Programming

Programming with MIDlet is very much similar to creating a J2SE application. However, a MIDlet is less robust than a J2SE application as there exist a lot of restrictions, imposed by the small computing devices.

Let's illustrate the details of developing a simplest MIDlet program i.e. A Hello World program in the Java platform ME Style.

A MIDlet is a java class which extends the MIDlet class. It acts as an interface between an application statements and the run-time environment (controlled by the application Manager). A MIDlet class should contain 3 abstract methods that are used to manage the life cycle of the MIDlet by the application manager. These abstract methods are:

- startApp()**,
- pauseApp()**,
- destroyApp()**.

**startApp()**

## Java ME

The `startApp()` method is called by the application manager as soon as the MIDlet is started. It contains statements that are executed each time the application begins execution.

### **pauseApp()**

The `pauseApp()` method is called before the application manager temporarily stops the MIDlet. The application manager restarts the MIDlet by recalling the `startApp()` method.

### **destroyApp()**

The `destroyApp()` method is called prior to the termination of the MIDlet by the application manager.

Lets see the basic shell of a MIDlet through a example.

**In our example**, the MIDlet class called `BasicMIDletShell` extends the MIDlet class. Any name can be used for a class as long as it conforms to the Java class naming conventions.

```
public class BasicMIDletShell extends MIDlet
{
public void startApp()
{
}
public void pauseApp()
{
}
public void destroyApp( boolean
unconditional)
{
}
```

### **MIDP 2 Game Classes**

The popularity of Java platform ME and game development has sprouted several carrier and manufacturer specific custom classes supporting the game development. However, the main problem with this is portability, e.g. using Siemens Sprite class make

it difficult for the user to port the game to a Nokia handset, as it requires to re-implement the sprite class.

**Release of MIDP 2.0** removed some of these common problems occurred with the game portability. MIDP 2.0 is released with the introduction of five new classes:

**\_ GameCanvas**

**\_ Sprite**

**\_ Layer**

**\_ LayerManager**

**\_ TiledLayer**

With these new game classes the user's code potentially become a lot easier issue and now you do not have to implement the custom classes such as Sprite. These classes are now a part of the underlying Java environment on the mobile handset.

# JBoss Seam : Stitching JSF and EJB3

Open source, and developed under the auspices of JBoss, Seam is a component framework that focuses to deliver full-featured JEE 5 applications in a lightweight code base i.e. requiring only a fraction of the code of regular JEE applications.

JBoss Seam is a robust JEE 5 framework that replaces the traditional way to develop the applications with some clever architecture and techniques. With Seam, JEE 5 and EJB3, seems to go a long way with their focus on the lightweight Java support and reducing the code bloat issues, which has been haunting the developers for a long time.

Although the specifications for JEE 5 are still not yet finalized, there is already a framework available. Seam being built on top of JEE 5 and EJB3 tries to further reduce the code required to build a functional application. Seam tries to put the new design patterns to resolve out the evergreen problems that haunt the development of the J2EE 1.4 applications, especially those concerned with the web-based user interface and demanding the EJBs at the back-end. This required a lot of tedious coding. Though most of the IDE tools tries to reduce the the developer's role in the code generation especially while coding the EJB interfaces and the required helper classes, still the developer faces the huge code base to manage and maintain.

Let's give a quick glance to the J2EE application development, and see how JBoss Seam proves to be a natural progression beyond JEE 5.

## The classic framework

To get familiar with the functionality of the JBoss Seam, it is vital to overview the role of the classic application framework, which orients around the two distinct phases:

- 1 creation of the user interface
- 2 creation of the application logic

Together with JEE 5, JBoss Seam attempts to simplify these two phases. Always the

business logic code is required to the develop the code individually and it can't be automated. Though, most of the application's tedious code orients around creating the UI to handle the user interactions and to provide the support for the code to arrange data back and forth between UI components and the application components. Seam has made a superb attempt to eliminate the bulkiness of such required codes. **Figure 1** illustrates the classic framework. Much of this classic architecture still applies today – even to JEE 5 and JBoss Seam.

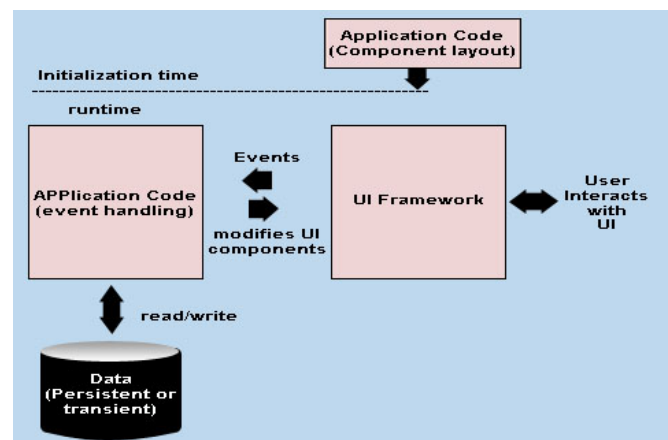


Figure 1: The classic framework

**Figure 1** depicts the classic UI framework along with the application flow. The UI framework provides the components that allows the developers to easily create and maintain a layout in their initialization code. At the runtime, this UI framework constructs the GUIs out of these components, and allows the user to interact with the system. At well-defined points of the user interaction, events are fired from the application components. Being an application developer, you just need to write the code to handle these events. This is sometimes referred as the "events-driven programming".

## Java Server Faces and JEE 5

Officially sun is going to include Java Server Faces (JSF) as its built-in UI framework with the release of its upcoming JEE 5 specifications. JSF has implemented the classic UI features as its UI framework architecture to meet the presentation purpose. However it needs to



# JBoss Seam : Stitching JSF and EJB3

meet the runtime challenges over a network to provide the actual user interface i.e. a web-based user interface. **Figure 1**

There has been made great efforts to provide the componentized user interfaces over the web to reduce the response time in reloading a changed UI. However to present a componentized user interface over the web is still challenging as it demands the framework to utilize the stateless HTTP protocol to render the changing UIs. However, every time the end user gets each UI as a separate dynamically generated web page and every time it requires the web browsers to interact with the UIs. Though, stateless HTTP protocol forces the actual rendered UI to stay loosely coupled with the code generating those UI. Classic GUI APIs like Swing demands a tightly coupling of the application code with the GUI. However, Unlike Swing, JSF decouples the presentation code at the expense of sending a completely new web page over the network.

Figure 2 illustrates one of the core features of the JSF framework i.e. loose coupling between the networked UI framework and the code that renders the UI. JSF provides a flexibility to use a number of render kits to render different user interfaces. For instance - the most frequently used render kit is the HTML render kit.

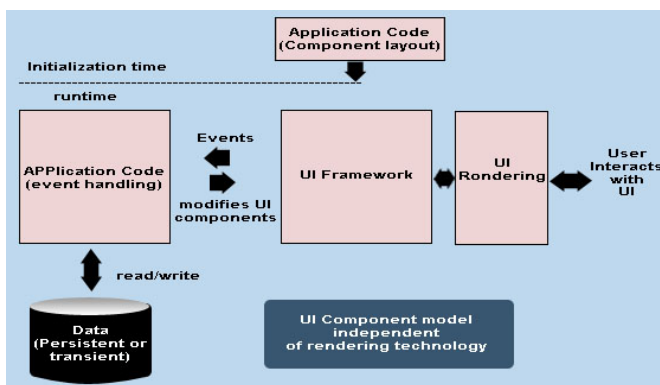


Figure 2: The rendering framework

In **Figure 2**, notice the gradual change made to the classic UI architecture to get the new rendering framework. Only there has been introduced a little change in writing the application code. Still the application handles

the UI component events, co-ordinates the access of data and modifies the state of UI components.

On the other hand, there is a little observable difference from the application developer's perspective that lies between classic tightly coupled GUI frameworks and the contemporary loosely-coupled web-based component UI frameworks.

However with the release of JEE 5 – It will be simpler to create the JEE applications almost as simple as coding the Swing applications. In JEE 5, Most of the concerned changes with the JSF specifications will orient around to provide a rich set of UI components via JSP presentation technology to create highly flexible web-based user interfaces.

## JSF: Facing Code-bloat issue

Achieving the desirable illusion of tightly coupled UI components cause the application developers to preserve the undesirable side-effect of code bloat when the JSF works along with JEE. Most of the typical JEE 5 applications developed with JSF and JSP perform few tasks repeatedly i.e. executing a code performing the tasks – again and again. There are few of the tasks that are retrieved as a part of Model section of the developed MVC application using different UI, few of such frequently occurring tasks are:

- to create backing beans that bind to JSF UI components
- to create the application objects for application logic
- to shuffle the data between the backing beans and the application objects
- to co-ordinate the data transfer between the JSF UI components and the backing beans
- to persist the application objects, perhaps through yet another set of objects used for persistence

**Figure 3** illustrates some of these interactions. The thin arrows illustrate the source of the

# JBoss Seam : Stitching JSF and EJB3

tedious code often found in the JEE and JSF coding.

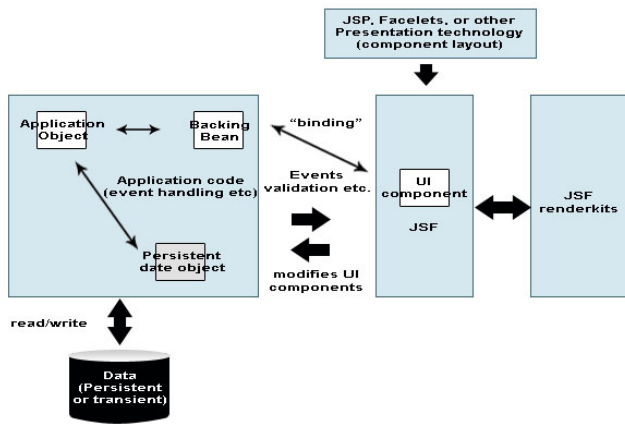


Figure 3: JSF and JSP

As the number of UI elements increases, the amount of the code performing the tedious task also increases. Much of the tedious code is generated as a result of the deficiency lying with the inefficient combination of the JEE and JSF. Such deficiencies are concerned with the issues like the UI components are not directly usable as application objects, similarly the application objects are not enough capable to persist information.

Additionally, JSF applications can not completely escape from the stateless nature of the HTTP protocol. Still it needs the alternative mechanisms, such as cookies or HTTP sessions to maintain the application state between more than one web page. This adds as a major contributor leading to the JEE code-bloat.

## Lets Explore the role of JBoss Seam in reducing the coding surface

**Figure 4** shows how JBoss Seam reduces the potential coding surface. Apart from the component layout code, Seam enables the developers just to code the needed business logic.

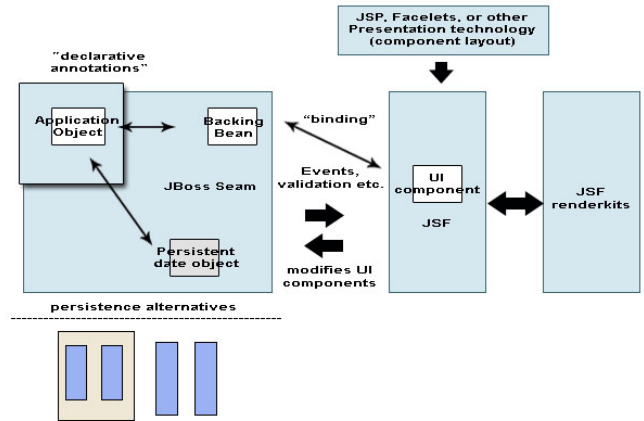


Figure 4: How JBoss Seam fights code-bloat

Seam achieves the code reduction via employing the modern framework techniques like to automate the code generation process( based on compile time inspection of the application objects), to intercept dynamically and to provide the behavioral modification at runtime. Much of what JBoss Seam does is completely invisible to the application developers however still it requires the developers to know a lot about the JEE 5 runtime environment and JSF in particular.

## Systematic code reduction with Seam

JBoss Seam has eliminated a lot of coding part that developers need to support previously. Seam exactly identified the earlier deficiencies lying with the JEE and JSF combination. Now seam has reduced the task of developers, they no longer have to create the JSF backing beans and write code to coordinate the transferring data between JSF components and the backing beans. Seam generates the JSF backing beans and coordinates the data shuttling between these beans and the application object. Even the developers no longer need to write the object persistence code for their application objects as Seam itself generates and manages them. Moreover, Seam developers can even :

- 1 use their application object as their UI object

# JBoss Seam : Stitching JSF and EJB3

2 use their application object as their persistent object

Now, the developers only need to create the set of core objects i.e. application objects.

JBoss Seam generates and manages a huge amount of code itself to flexibly control the behaviour of the entire code at the run times. Seam achieves this feature through a very rich set of annotations as Seam is consistent with the heavy use of annotations in JEE 5. Annotations are there to declaratively control the things like how to generate and manage the application code. Though the code generated and managed by JBoss Seam is not as flexible as tedious hand-crafted code written by the developers. But then, if we consider the amount of the code that the developers no more need to write, then the trade-off in robustness and maintainability may justify the reduced flexibility.

## Seam with new conversational context

JBoss Seam has highly reduced the tedious coding required to manage the states between the discrete UI pages. Seam achieves this by providing a conversational context for maintaining application level attributes. Seam allows a conversation to maintain over many HTTPconnections.

Now with seam application developers just need to declare an application object to maintain it in the conversation scope and Seam ensures to make the instance available throughout the conversation. It just requires to specify - when a conversation begins and when it should end.

Conversation implementation is a nice feature added to the JBoss Seam. This provides the ability to track and manage multiple concurrent conversations from the same user.

## Seam's Support for a myriad of persistence alternatives

Figure 4 illustrates the role of JBoss Seam

in managing persistence. Just it requires to add the appropriate annotations to the application object, and further Seam co-ordinates the required generation of code and the runtime interceptions.

For the actual persistence work Seam either uses EJB 3 CMP with a support in JEE 5 (i.e. currently known as Glassfish Server) or the most popular ORM i.e. Hibernate.

Figure 5 illustrates the most suitable persistence alternatives for JBoss Seam i.e. – JBoss Seam can use EJB 3 CMP to persist the application object.

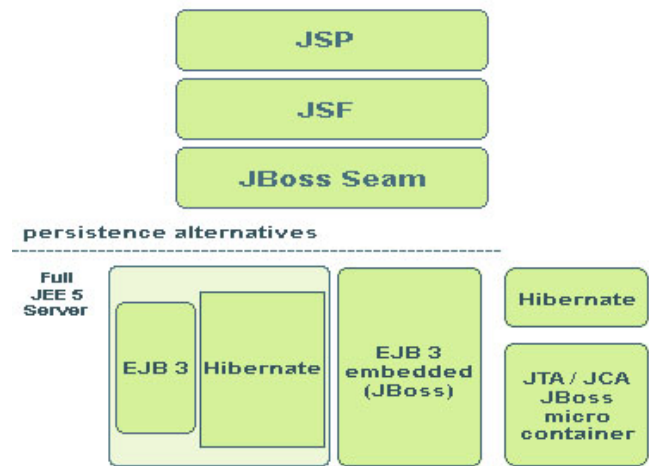


Figure 5: Persistence alternatives for JBoss Seam

Now with Seam, Developers need not to develop any additional code to support this. Just they can use an entity bean for their application objects. This style of persistence is perfect for application running inside a JEE 5 container – such as the JBoss Application Server.

## Seam's flexibility to adapt the light-weight development environment

Now with Jboss Seam you can keep your application lightweight, and can rather do without any EJBs, as the Seam provides the

## JBoss Seam : Stitching JSF and EJB3

developers with an EJB-free solution. Developers can use Hibernate to persist the application objects. JBoss Seam supports transactions with Hibernate when an application runs inside a JEE 5 container. Even the developers can use Jboss's configurable stand-alone embeddable EJB 3 container, which is highly compatible with JBoss Seam in providing the run time support to the applications. Now the developers have chances to use Hibernate without worrying to run the application inside a full-blown JEE 5 server, as they have the flexibility to use the JBoss microcontainer along with its transactional support. And it still run in a Java SE environment. This brings the light-weightness to the development environment.

This provides a set of diverse variety to the developers along with the multiple choices enabling it to adapt to different production scenarios. Once tested, the application code does not need to be modified while moving from one environment to the next. It requires only the minor changes in the configuration.



The advertisement features a blue background with white and yellow text. At the top, it says "Grow your business with our SERVICES" in a large, bold font. Below this, a list of services is presented with horizontal lines pointing to the right. At the bottom, the RoseIndia logo is displayed, consisting of a stylized orange and white bird-like shape, followed by the text "RoseIndia" and the URL "http://www.roseindia.net/services/".

Grow your  
business  
with our  
**SERVICES**

- Software Solutions
- E-Commerce Solutions
- Website Development
- Web Promotion
- Web Hosting
- Content Development

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



# Maven 2 Plug-in

Plugins are used to interact with a host application to provide a specific task on demand. Maven provides a plugin execution framework and rest of the work is done by the plugins i.e. real action is performed by the plugins like compiling code, creating jar files, creating war files, testing the code, creating project documentation etc. For Instance... **Clean** is one of the simplest maven 2.0 plugins available. When we run "mvn clean", the "clean" goal is executed and the target directory is removed.

Here, we are providing a list of core and other plugins below:

<b>Plugin</b>	<b>Description</b>
Core plugins	Corresponding to default core phases
clean	Clean up after the build.
compiler	Compiles Java sources.
deploy	Deploy the built artifact to the remote repository.
install	Install the built artifact into the local repository.
resources	Copy the resources to the output directory for including in the JAR.
site	Generate a site for the current project.
surefire	Run the Junit tests in an isolated classloader.
verifier	Verifies the existence of certain conditions.

## **Packaging types Related to packaging / tools respective artifact types.**

ear	Generate an EAR from the current project.
ejb	Build an EJB from the current project.
jar	Build a JAR from the current project.
rar	Build a RAR from the current project.
war	Build a WAR from the current project.

## **Reporting**

changelog

changes

checkstyle  
clover  
doap

docck

javadoc

jxr

pmd  
project-info-reports  
surefire-report

## **Related to generating reports**

Generate a list of recent changes from your SCM.

Generate a report from issue tracking or a change document.

Generate a checkstyle report.  
Generate a Clover report.  
Generate a DOAP file from a POM.

Documentation checker plugin.

Generate Javadoc for the project.

Generate a source cross reference.

Generate a PMD report.

Generate a standard project reports.

Generate a report based on the results of unit tests.

## **Tools**

## **Tools available through Maven by default**

ant

Generate an Ant build file for the project.

antrun

Run a set of ant tasks from a phase of the build.

archetype

Generate a skeleton project structure from an archetype.  
assembly Build an assembly of sources and/or binaries.

dependency

Dependency manipulation and analysis.

enforcer

Environmental constraint checking, User Custom Rule Execution.

gpg

Create signatures for the artifacts and poms.

help

Get information about the working environment for the project.

invoker


Run a set of Maven projects and verify the output

one

A plugin for interacting with legacy Maven 1.x repositories


## Maven2 Plug-in

	and builds.
patch	Use the gnu patch tool to apply / patch files to source code.
plugin	Create a Maven plugin descriptor for any Mojo's found in the source tree, to include in the JAR.
release	Release the current project remote-resources Copy remote resources to the output directory for inclusion in the artifact.
repository	Plugin to help with repository-based tasks.
scm	Generate a SCM for the current project.
source	Build a JAR of sources for use in IDEs and distribution to the repository.
<b>IDEs</b>	<b>Plugins that simplify integration with IDEs</b>
eclipse	Generate an Eclipse project file for the current project.
idea	Create/update an IDEA workspace for the current project.



**8 × 3 = 24hours**

**It's always for 24h**



**NEWSTRACK india**

# Tomahawk Tags

Tomahawk tags are the collection of standard components with extended functionality and supports all the existing JSF components with additional sets of functionality. Some tomahawk tags are described in the subsequent sections.

## 1. Tomahawk document tag

This tag is used to embed whole document into the jsp page. It is equivalent to the HTML `<html>` tag. We can use this tag in place of `<html>` tag in our JSP page. It has only one attribute "state" that is used to specify the state stored by this component.

### Code Description :

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<f:view>
  <t:document>
  <head>
  <meta http-equiv="Content-Type"
    content="text/html; charset=iso-8859-1">
  <title>t:document example</title>
  </head>
  <body >
  <h:form>
  <t:outputText value="The document tag
is equivalent to HTML <html> tag."/>
  </h:form>
  </body>
  </t:document>
</f:view>
```

## Rendered Output:



The document tag is equivalent to HTML `<html>` tag.

## 2. Tomahawk documentHead tag

This tag is used to encapsulate the head of the document. It is equivalent to the HTML `<head>` tag. We can use this tag in place of `<head>` tag in our JSP page. It has only one attribute "state" that is used to specify the state stored by this component.

### Code Description :

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<f:view>
  <t:document>
  <t:documentHead>
  <meta http-equiv="Content-Type"
    content="text/html; charset=iso-8859-1">
  <title>t:documentHead example</title>
  </t:documentHead>
  <body >
  <h:form>
  <t:outputText value="The documentHead tag
is equivalent to HTML <head> tag."/>
  </h:form>
  </body>
  </t:document>
</f:view>
```

# Tomahawk Tags

## Rendered Output:



The documentHead tag is equivalent to HTML `<head>` tag.



## 3. Tomahawk documentBody tag

This tag is used to encapsulate the body of the document. It is equivalent to the HTML `<body>` tag. We can use this tag in place of `<body>` tag in our JSP page. It has one attribute "state" that is used to specify the state stored by this component.

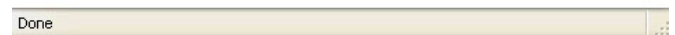
### Code Description:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<f:view>
<t:document>
<t:documentHead>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>t:documentBody example</title>
</t:documentHead>
<t:documentBody >
<h:form>
<t:outputText value="The documentBody tag is equivalent to HTML <body> tag."/>
</h:form>
</t:documentBody>
</t:document>
</f:view>
```

## Rendered Output:



The documentBody tag is equivalent to HTML `<body>` tag.



## 4. Tomahawk saveState tag

This tag is useful in persisting the backing bean and its properties longer than request scope but shorter than session scope by saving the state with the component tree. Traditionally, state is saved with the help of HttpSession object. MyFaces works differently without the use of HttpSession object. All state information of the current view and the model beans are encoded automatically with the client response and get restored at the next client request. If you want to save the whole bean then it can also be done with this tag just specifying the name of bean in EL expression in the value attribute. Same steps are followed in saving and restoring the whole bean as in the case of properties of bean. If you want the value of a bean property or the bean itself to be able to get saved and restored, it must implement the **Serializable** interface.

### Code Description :

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<f:view>
<t:document>
<t:documentHead>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

# Tomahawk Tags

```
<title>t.saveState example</title>
</t:documentHead>
<t:documentBody >
<t:saveState id="state1"
value="#{Bean.text1}"/>
<t:saveState id="state2"
value="#{Bean.text2}"/>
<t:saveState id="state3"
value="#{NextBean}"/>
<h:form>
<t:outputText value="Title1"/>
<t:inputText id="it1" value="#{Bean.text1}"
/></p>
<t:outputText value="Title2"/>
<t:inputText id="it2" value="#{Bean.text2}"
/>
</h:form>
</t:documentBody>
</t:document>
</f:view>
```

## 5. Tomahawk inputDate tag

This tag is useful in creating the component to input the date. This component can be of various types. We can create this component to input time only or date only or both. Time can also be supplied in 12 hour or 24 hour format. The time can also be set according to the supplied time zone. The popup calendar can also be rendered if needed. We can also use CSS and java script to make it useful according to our requirement.

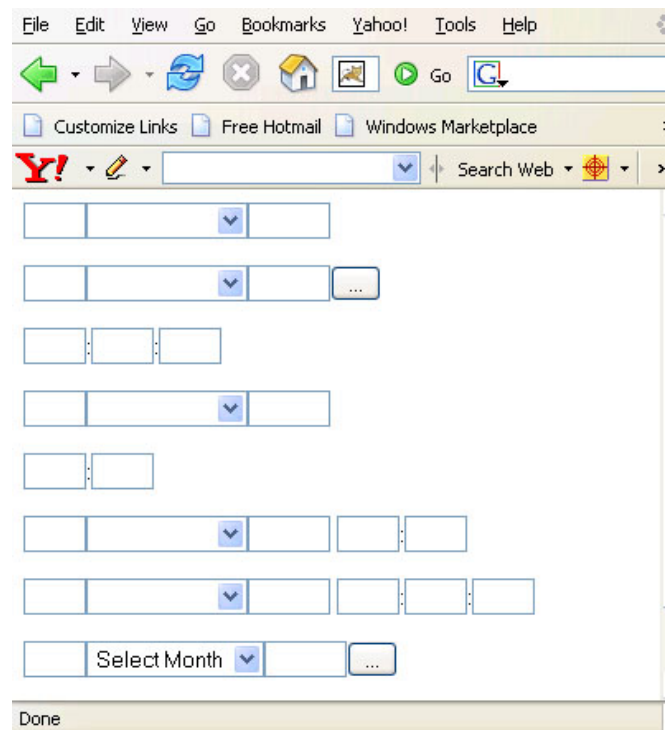
### Code Description :

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<f:view>
<t:document>
<t:documentHead>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<title>t.inputDate example</title>
<style type="text/css">
<!--
.highlight { background-color: #A8D1E8;
```

```
color:blue;}
—>
</style>
</t:documentHead>
<t:documentBody >
<h:form>
<t:inputDate id="date1" /></p>
<t:inputDate id="date2" opupCalendar="true"
/> </p>
<t:inputDate id="date3" type="time"/></p>
<t:inputDate id="date4" type="date"/></p>
<t:inputDate id="date5" type="short_time"
/></p>
<t:inputDate id="date6" type="both"/></p>
<t:inputDate id="date7" type="full"/></p>
<t:inputDate id="date8"
emptyMonthSelection="Select Month"
type="date" popupCalendar="true"
onmouseover="this.className='highlight'"
onmouseout="this.className='normal'"/>
</p>
<t:inputDate id="date9" type="time"
ampm="true"
emptyAmpmSelection="AM/PM" />
</h:form>
</t:documentBody>
</t:document>
</f:view>
```

### Rendered Output





# Tomahawk Tags

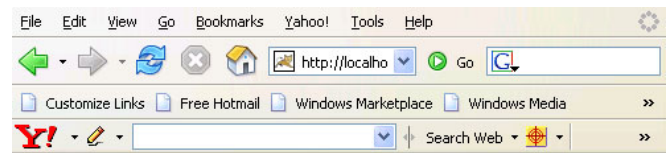
## 6. Tomahawk htmlTag tag

This tag is used to use html tag for its child component. This provides "value" attribute to specify the name of the html tag to be used. For example, if we want to make any string as bold then enclose it within <t:html> tag and set its value as "b" because in html to make any character bold we use <b> tag. If we don't provide any value to the value attribute then all its child component are rendered but no html tag is applied on them.

### Code Description :

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<f:view>
<t:document>
<t:documentHead>
<meta http-equiv="Content-Type"
content="text/html;
charset=iso-8859-1">
<title>t:htmlTag example</title>
</t:documentHead>
<t:documentBody >
<h:form>
<t:htmlTag value="h1">t:htmlTag Example
</t:htmlTag>
<t:htmlTag value="b"><i>Bold & Italic font
</i></t:htmlTag>
<t:htmlTag value="br"/>
<t:htmlTag value="b" rendered="false" >
Bold property will not be applied here.
</t:htmlTag>
</h:form>
</t:documentBody>
</t:document>
</f:view>
```

## Rendered Output :



### t:htmlTag Example

***Bold & Italic font***

Bold property will not be applied here.

## 7. Tomahawk dataList tag

This tag is like dataTable tag but the difference between the two is that it does not render a table. In this tag the data rows are controlled and rendered by the use of "layout" attribute. It supports three layouts "simple", "unorderedList", "orderedList". The default value is simple. When "simple" is used the items are rendered normally, when "unorderedList" is used the list is rendered as an HTML unordered list and in the case of "orderedList", the list is rendered as an HTML ordered list.

### Code Description :

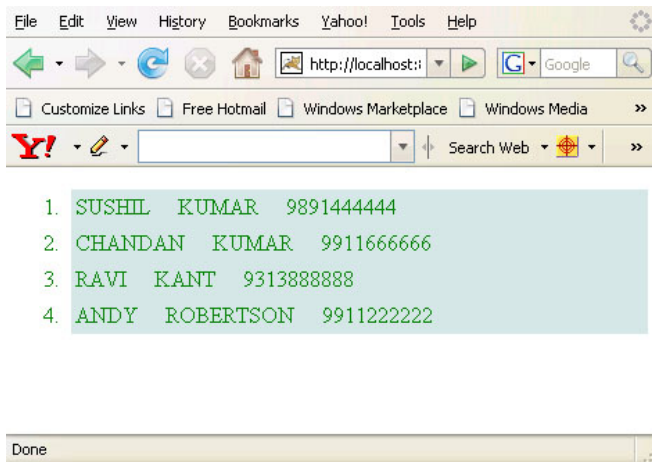
```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>

<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<title>t:dataList example</title>
<style type="text/css">
<!--
.dataListStyle {
color: green;
background-color: #D0E6E0;
padding: 3;
}
-->
</style>
```



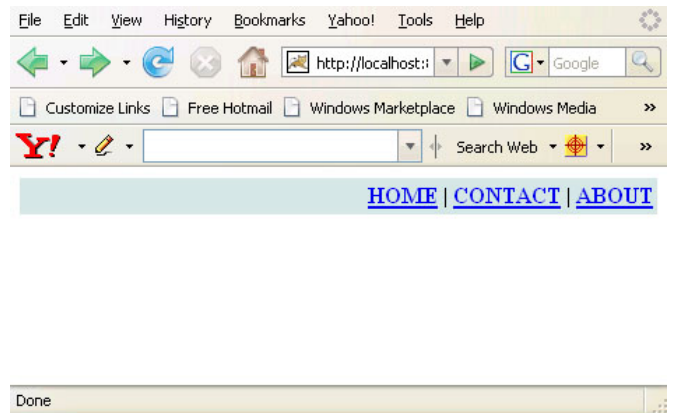
# Tomahawk Tags

## Rendered Output:



```
<f:view>
<h:form id="form1" >
<t:div id="div1" styleClass="divStyle">
<t:commandLink value="HOME"
action="welcome"/> |
<t:commandLink value="CONTACT"
action="welcome"/> |
<t:commandLink value="ABOUT"
action="welcome"/>
</t:div>
</h:form>
</f:view>
</body>
</html>
```

## Rendered Output:



## 8. Tomahawk div tag

This tag is used to place an html div around its children. So instead of using html div tag we can use JSF tomahawk's own div tag. In this example, div tag uses style class "divStyle" that will be effective on the particular area captured by div tag.

## Code Description:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<title>t:div example</title>
<style type="text/css">
<!--
.divStyle {
text-align: right;
background-color: #D0E6E0;
padding: 3;
font-weight:bold;
}
-->
</style>
</head>
<body >
```

# Struts2

## Apache Struts: A brief Introduction

Apache Struts is an open-source framework used to develop Java web application. Originally developed by the programmer and author Craig R. McClanahan was later taken over by the Apache Software Foundation in 2002. Struts have provided an excellent framework for developing application easily by organizing JSP and Servlet and basic java code. Struts1 with all standard Java technologies and packages of Jakarta assists in creating an extensible development environment. However, with the growing demand of web applications, Struts 1 does not stand firm and needs to be changed with the increasing demand. This led to the creation of Struts2, which are more users friendly with the features like Ajax, rapid development and extensibility.

Struts is a well-organized framework based on MVC architecture. In Model-View-Controller Architecture, Model refers to the business or database code, the View represents the UI design code and the Controller refers to the navigational code. All these together makes Struts an essential framework for building Java application. But with the development of new and lightweight MVC based frameworks like Spring, Stripes and Tapestry, it becomes necessary to modify the Struts framework. So, the team of Apache Struts and another J2EE framework, WebWork of OpenSymphony joined hand together to develop an advanced framework with all possible developing features that will make it developer and user friendly.

Struts 2 has combined features of Struts 1.x and WebWork 2 projects that advocates higher level application by using the architecture of WebWork2 with the features including a plugin framework, a new API, Ajax tags etc. So the Struts communities and the WebWork team brought together several special features in WebWork2 to make it more advance in the Open Source world. Later the name of WebWork2 has changed to Struts2. Hence, Apache Struts 2 is a dynamic, extensible framework for a complete application development from building, deploying and maintaining.

## Why Struts 2

The new version Struts 2.0 is a combination of the Struts action framework and Webwork. According to the Struts 2.0.1 release announcement, some key features are: Why Struts 2

- 1 Simplified Design** - Programming the abstract classes instead of interfaces is one of design problem of struts1 framework that has been resolved in the struts 2 framework. Most of the Struts 2 classes are based on interfaces and most of its core interfaces are HTTP independent. Struts 2 Action classes are framework independent and are simplified to look as simple POJOs. Framework components are tried to keep loosely coupled.
- 2 Simplified Actions** - Actions are simple POJOs. Any java class with execute() method can be used as an Action class. Even we don't need to implement interfaces always. Inversion of Control is introduced while developing the action classes. This makes the actions to be neutral to the underlying framework.
- 3 No more ActionForms** - ActionForms feature is no more known to the struts2 framework. Simple JavaBean flavored actions are used to put properties directly. No need to use all String properties.
- 4 Simplified testability** - Struts 2 Actions are HTTP independent and framework neutral. This enables to test struts applications very easily without resorting to mock objects.
- 5 Intelligent Defaults** - Most configuration elements have a default value which can be set according to the need. Even there are xml-based default configuration files that can be overridden according to the need.

# Struts2

**6 Improved results** - Unlike ActionForwards, Struts 2 Results provides flexibility to create multiple type of outputs and in actual it helps to prepare the response.

**7 Better Tag features** - Struts 2 tags enables to add style sheet-driven markup capabilities, so that we can create consistent pages with less code. Struts 2 tags are more capable and result oriented. Struts 2-tag markup can be altered by changing an underlying stylesheet. Individual tag markup can be changed by editing a FreeMarker template. Both JSP and FreeMarker tags are fully supported.

**8 Annotations introduced:** Applications in struts 2 can use use Java 5 annotations as an alternative to XML and Java properties configuration. Annotations minimize the use of xml.

**9 Stateful Checkboxes** - Struts 2 checkboxes do not require special handling for false values.

**10 QuickStart** - Many changes can be made on the fly without restarting a web container.

**11 Customizing controller** - Struts 1 lets to customize the request processor per module, Struts 2 lets to customize the request handling per action, if desired.

**12 Easy Spring integration** - Struts 2 Actions are Spring-aware. Just needs to add Spring beans.

**13 Easy plugins** - Struts 2 extensions can be added by dropping in a JAR. No manual configuration required.

**14 AJAX support** - The AJAX theme gives interactive applications a significant boost.

The framework provides a set of tags to help you ajaxify your applications, even on Dojo.

## The AJAX features include:

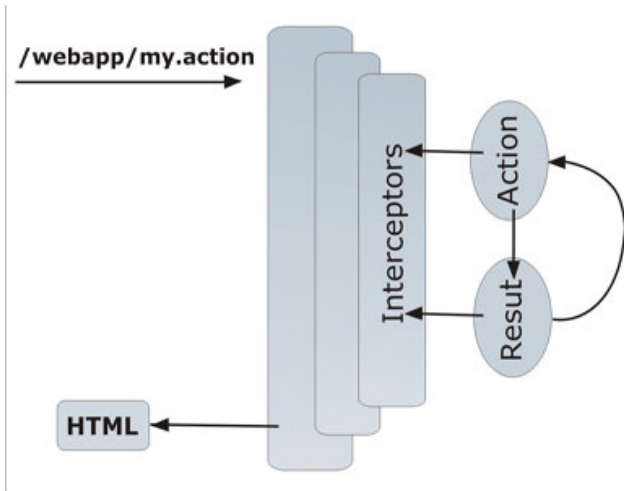
- AJAX Client Side Validation
- Remote form submission support (works with the submit tag as well)
- An advanced div template that provides dynamic reloading of partial HTML
- An advanced a template that provides the ability to load and evaluate JavaScript remotely
- An AJAX-only tabbed Panel implementation
- A rich pub-sub event model
- Interactive auto complete tag
- Request Lifecycle in Struts 2 applications

## Request for a resource by the user is processed in the sequence given below:

- 1 User Sends request:** User sends a request to the server for some resource.
- 2 FilterDispatcher determines the appropriate action:** The FilterDispatcher looks at the request and then determines the appropriate Action.
- 3 Interceptors are applied:** Interceptors configured for applying the common functionalities such as workflow, validation, file upload etc. are automatically applied to the request.
- 4 Execution of Action:** Then the action method is executed to perform the database related operations like storing or retrieving the data from database.
- 5 Output rendering:** Then the Result renders the output.
- 6 Return of Request:** Then the request returns through the interceptors in the reverse order. The returning request allows us to perform the clean-up or additional processing.
- 7 Display the result to user:** Finally the control is returned to the servlet container, which sends the output to the user browser.

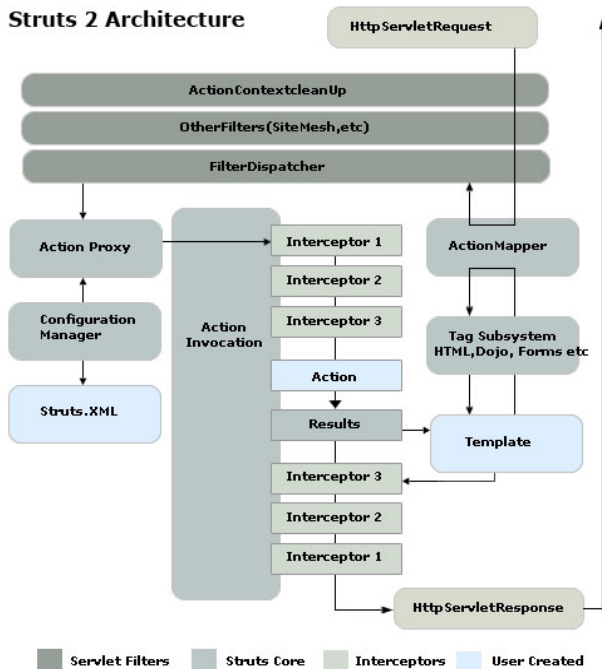


# Struts2



## Struts 2 Architecture

The following diagram depicts the architecture of Struts 2 Framework. Following diagram shows that the initial request goes to the servlet container, which is then passed through standard filter chain.



## The filter chain includes:

- 1 Action ContextCleanUp filter:** The ActionContextCleanUp filter is optional and it is useful when integration has to be done with other technologies like SiteMash Plugin.
- 2 FilterDispatcher:** Next the FilterDispatch is called, which in turn uses the ActionMapper to determine weather to invoke an Action. If the action is required to be invoked, the FilterDispatcher delegates the control to the ActionProxy.
- 3 ActionProxy:** The ActionProxy takes the help from Configuration Files manager, which is initialized from the struts.xml. Then the ActionProxy creates an ActionInvocation, which implements the command pattern. The ActionInvocation process invokes the Interceptors (if configured) and then invokes the action. The ActionInvocation looks for proper result. Then the result is executed, which involves the rendering of JSP or templates.

Then the Interceptors are executed again in reverse order. Finally the response returns through the filters configured in web.xml file. If the ActionContextCleanUp filter is configured, the FilterDispatcher does not clean the ThreadLocal ActionContext. If the ActionContextCleanUp filter is not present then the FilterDispatcher will cleanup all the ThreadLocals present.

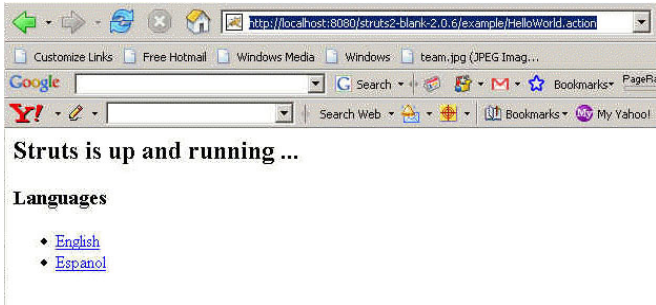
Download Struts 2.0 and install Blank application on the Tomcat server

Download the Struts 2.0 at <http://struts.apache.org/download.cgi>. Extract the downloaded struts distribution **struts-2.0.6-all.zip** into your favorite directory.

To install the struts blank application copy "**struts2-blank-2.0.6**" from "<extracted directory>\struts-2.0.6-all\struts-2.0.6\apps" into the webapps folder of the tomcat directory.

# Struts2

Tomcat will automatically deploy the application. To test the struts-blank application type <http://localhost:8080/struts2-blank-2.0.6> in the browser and the struts-blank application get displayed in your browser window.



## Struts 2 Hello World - Developing Hello World Application

In this section we will develop Hello World based on the Struts 2 Framework. Our Struts 2 Hello World application is the first step towards developing applications based on the Struts 2 Framework. It covers the basic steps like creating directory structure, developing build.xml file to build the application using ant build tool. Then we will write JSP, Java and configuration files required for the application.

Creating directory structure and Ant build file for the project

**Step1:** Extract struts 2 download and copy struts2-blank-2.0.6.war (If you are using latest version of struts 2 then it may be different for you) to your tomcat's webapps directory. Rename struts2-blank-2.0.6 to struts2example and unzip it in the tomcat's webapps directory. Start tomcat and type <http://localhost:8080/struts2example/> into your browser. Now, you have successfully installed struts 2 blank application to start with.

**Step 2:** Now delete the content of struts2example\WEB-INF\src and struts2example\WEB-INF\classes directories, as we don't need these files that comes with struts 2 blank application.

**Step 3:** Create build.xml file in the struts2example\WEB-INF\src and paste the following content in the build.xml file.

```
<project name="struts2example" basedir=".."
  default="all">
  <!-- Project settings -->
  <property name="project.title"
    value="JavaJazzUp Struts 2"/>
  <property name="project.jar.file"
    value="struts2example.jar"/>
  <path id="class.path">
  <fileset dir="lib">
  <include name="**/*.jar"/>
  </fileset>
  <fileset dir="libext">
  <include name="**/*.jar"/>
  </fileset>
  </path>
  <!-- Classpath for Project -->
  <path id="compile.classpath">
  <pathelement path="lib/commons-
    digester.jar"/>
  <pathelement path="lib/commons-
    digester.jar"/>
  <pathelement path="lib/struts.jar"/>
  <pathelement path="libext/servlet-api.jar"/>
  <pathelement path="libext/catalina-ant.jar"
  />
  <pathelement path="classes"/>
  <pathelement path="${classpath}"/>
  </path>
  <!-- Check timestamp on files -->
  <target name="prepare">
  <tstamp/>
  <copy file="src/struts.xml"
    todir="src/classes"/>
  </target>
  <!-- Copy any resource or configuration files
  -->
  <target name="resources">
  <copy todir="src/classes"
    includeEmptyDirs="no">
  <fileset dir="src/java">
  <patternset>
  <include name="**/*.conf"/>
  <include name="**/*.properties"/>
  <include name="**/*.xml"/>
  </patternset>
  </fileset>
  </copy>
  </target>
```

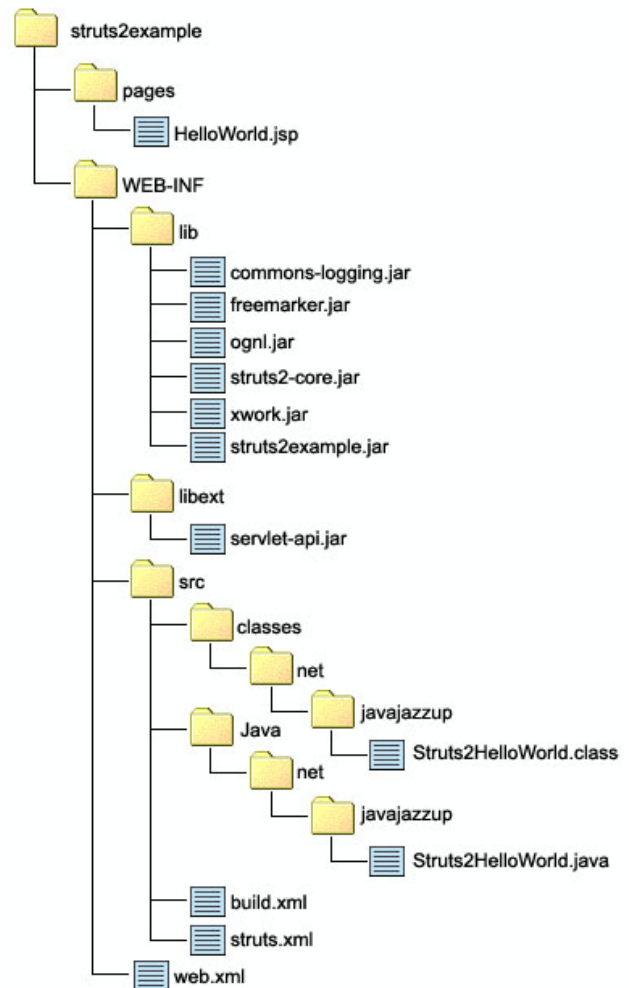
## Struts2

```
<!-- Normal build of application -->
<target name="compile"
depends="prepare,resources">
<javac srcdir="src" destdir="src/classes"
  debug="true"
debuglevel="lines,vars,source">
<classpath refid="class.path"/>
</javac>
<jar jarfile="lib/${project.jar.file}"
basedir="src/classes"/>
</target>
<!-- Remove classes directory for clean build -->
<target name="clean" description="Prepare
for clean build">
<delete dir="classes"/>
<mkdir dir="classes"/>
</target>
<!-- Build Javadoc documentation -->
<target name="javadoc"
description="Generate JavaDoc API docs">
<delete dir="./doc/api"/>
<mkdir dir="./doc/api"/>
<javadoc sourcepath="./src/java" destdir="./
doc/api"
classpath="${servlet.jar}:${jdbc20ext.jar}"
packagenames="*" author="true"
private="true" version="true"
windowtitle="${project.title}API
Documentation"
doctitle="&lt;h1&gt;${project.title}Documentation
(Version ${project.version})&lt;/h1&gt;"
bottom="Copyright &#169; 2002">
<classpath refid="compile.classpath"/>
</javadoc>
</target>
<!-- Build entire project -->
<target name="project"
depends="clean,prepare,compile"/>
<!-- Create binary distribution -->
<target name="dist" description="Create
binary distribution">
<mkdir dir="${distpath.project}"/>
<jar jarfile="${distpath.project}/
${project.distname}.jar"
basedir="./classes"/>
<copy file="${distpath.project}/
${project.distname}.jar"
todir="${distpath.project}"/>
<war basedir="./"
warfile="${distpath.project}/
```

```
${project.distname}.war"webxml="web.xml">
<exclude name="${distpath.project}/
${project.distname}.war"/>
</war></target>
<!-- Build project and create distribution-->
<target name="all" depends="project"/>
</project>
```

**Step 4:** Create directory libext under the struts2example\WEB-INF\ and copy latest Servlets api jar (in our case servlet-api.jar) file over there. This library file will be used to compile Servlets in our application.

**Step 5:** Now create directories java and classes under struts2example\WEB-INF\src. The directory struts2example\WEB-INF\src\java will be used to put all the java sources file and directory struts2example\WEB-INF\src\classes will be used by ant build utility to store compiled java files.



# Struts2

## Writing JSP, Java and Configuration for Hello World Application

Now, we will write JSP, Java and required configuration files for our Struts 2 Hello World application. In struts 2, struts.xml is used to configure the applications. Our application is very simple application that displays Hello World message along with current date and time of the server. When the user requests for the resource then this request is sent to the struts framework. Then struts framework sends the input to the action class (in our case Struts2HelloWorld.java) and selects the resource "/pages/HelloWorld.jsp" to render as response to the user. In this example we have to develop three parts view, Action class and mapping (struts.xml) to couple action and page.

### Developing View:

This page is used to display the result on the browser. The HelloWorld.jsp is view part of our application. Create "HelloWorld.jsp" in the struts2example\pages directory and add the following content:

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 Hello World Application!
</title>
</head>
<body>
<h2><s:property value="message" /></h2>
<p>Current date and time is:
<b><s:property value="currentTime" /></b>
</body>
</html>
```

The line `<%@ taglib prefix="s" uri="/struts-tags" %>` declares data tag library of the struts. The struts data tag is used to display the dynamic data. The tag `<s:property value="message" />` and `<s:property value="currentTime" />` calls the methods `getMessage()` and `getCurrentTime()` respectively of the Struts2HelloWorld action class and merges the values with response. Developing

Action (to interact with Model):

Now create Struts2HelloWorld.java and save it to the "struts2example\WEB-INF\src\java\net\javajazzup" directory. This action class creates the message to be displayed on the screen. Here is the code of Struts2HelloWorld.java:

### Struts2HelloWorld.java

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

public class Struts2HelloWorld extends
ActionSupport {

public static final String MESSAGE = "Struts 2
Hello World";

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

private String message;
public void setMessage(String message){
this.message = message;
}

public String getMessage() {
return message;
}

public String getCurrentTime(){
return new Date().toString();
}
}
```

### Developing Controller Configuration File:

Struts 2 uses the struts.xml file for configuring the application. Create struts.xml file and save it in the "struts2example\WEB-INF\src" directory with the following content.

## Struts2

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts
Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-
2.0.dtd">

<struts>
<constant
name="struts.enable.DynamicMethodInvocation"
value="false" />
<constant name="struts.devMode"
value="true" />
<package name="javajazzup" namespace="/"
javajazzup " extends="struts-default">
<action name="HelloWorld"
class="net.javajazzup.Struts2HelloWorld">
<result>/pages/HelloWorld.jsp</result>
</action>
<!-- Add actions here -->
</package>
<!-- Add packages here -->
</struts>
```

The struts.xml file should be present in the class path of the application. You can either include it in the jar and place in the lib directory of the application or place it in the classes directory of the web application. In our application we are using ant build tool, which is including it in the jar file.

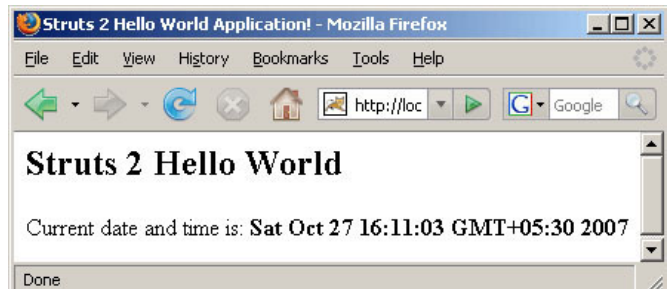
### Building the application

I am assuming that you have already installed ant build tool on your machine. Since we are using the ant build tool, the building the application is very easy. To build the application open command prompt and go to "struts2example\WEB-INF\src" directory of the web application and issue the "ant" command. The ant build tool will compile the java file and create jar file "struts2example.jar" into the lib directory of your web application. Here is the output of ant build tool:

```
C:\apache-tomcat-
5.5.23\webapps\struts2example\WEB-
INF\src>ant
Buildfile: build.xml
clean:
 [delete] Deleting directory C:\apache-
tomcat-5.5.23\webapps\struts2example\WE
B-INF\classes
 [mkdir] Created dir: C:\apache-tomcat-
5.5.23\webapps\struts2example\WEB-INF\
classes
prepare:
resources:
compile:
[javac] Compiling 1 source file to C:\apache-
tomcat-5.5.23\webapps\struts2ex
ample\WEB-INF\src\classes
[jar] Building jar: C:\apache-tomcat-
5.5.23\webapps\struts2example\WEB-INF
\lib\struts2example.jar
project:
all:
BUILD SUCCESSFUL
Total time: 4 seconds
C:\apache-tomcat-
5.5.23\webapps\struts2example\WEB-
INF\src>
```

### Testing the application

To test the application start the tomcat server and type http://localhost:8080/struts2example/javajazzup/HelloWorld.action in address bar and hit enter. It will display the following figure:





# Struts2

The application will display message "Struts 2 Hello World" along with current date and time of the server.

## How application works?

Here is the brief description on how Struts 2 Hello World Application works:

**1.** When the user sends a request for the url `http://localhost:8080/struts2example/javajazzup/HelloWorld.action`. The container requests for the resource "HelloWorld.action". By default web.xml file of struts blank application is configured to route all the request for \*.action through `org.apache.struts2.dispatcher.FilterDispatcher`. Here is the configuration from web.xml file:

```
<filter>
<filter-name>struts2</filter-name>
<filter-lass>
org.apache.struts2.dispatcher.FilterDispatcher
</filter-class>
</filter>
<filter-mapping>
<filter-name>struts2</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

**2.** Then the framework looks for the mapping for the action "HelloWorld" and then framework instantiates the appropriate class and calls the execute method. In this case action class is Struts2HelloWorld. Here is the configuration file from struts.xml, which defines the action mapping:

```
<action name="HelloWorld"
class="net.javajazzup.Struts2HelloWorld">
<result>/pages/HelloWorld.jsp</result>
</action>
```

**3.** Then the execute method sets the message and returns SUCCESS.

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

Then framework determines which page is to be loaded if SUCCESS is returned. In our case framework tells the container to load HelloWorld.jsp and render the output.

In the struts 2 framework, Actions are used to process the form and user request. The execute method of the action returns SUCCESS, ERROR, or INPUT value. Then based on these values framework tells the container to load and render the appropriate result.

**4.** Container processes the HelloWorld.jsp and generates the output.

The output in the HTML format is sent to the browser.

# Design Pattern

Structural Patterns are design patterns, which describe the best possible ways to combine the objects and classes forming a larger complex structure in an easy manner. It deals with the objects delegating responsibilities to other objects. It is a simple way to design and realize relationships between the entities.

In this article, we are discussing Decorator , Façade and Flyweight Design Patterns.

## I. Decorator Design Pattern

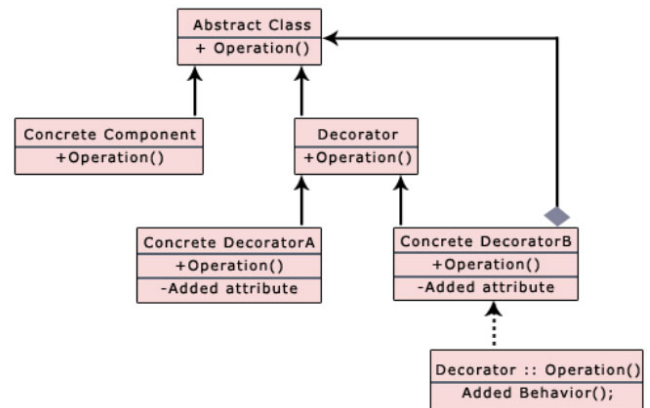
This section deeply discusses the Decorator design pattern, a kind of structural pattern.

In object-oriented programming, the decorator pattern allows the developers to add new behaviors to an existing functionality of an object, dynamically.

The decorator pattern can be used whenever there is a need to add some additional functionality to an object or to the group of objects. It provides a flexible alternative for subclassing to extend the functionality. This pattern is also known as the “**wrapper**” design pattern as it works by wrapping around the original object to get a new “decorator” object. Wrapping is typically achieved by passing the original object as a parameter to the constructor of the decorator, and then it is ready to implement the new functionality.

Decorators are very much similar to subclassing feature of an object-oriented programming. The only difference is that, Subclassing adds behavior at compile time whereas a decorator provides a new behavior at the runtime.

The UML diagram for this pattern in our scenario looks something like this:

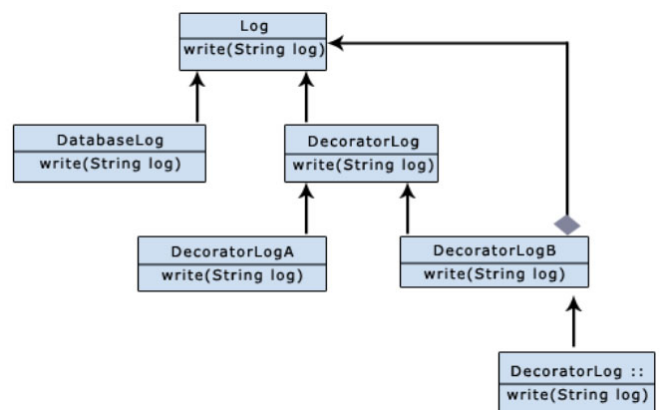


This UML diagram illustrates that the Decorators should be abstract classes and the concrete implementation should be derived from them according to the requirement.

A decorator can be added or removed from an object without realizing the client about any change that occurred. It is a good idea to use a Decorator in a situation where you want to change the behavior of an object repeatedly (by adding and subtracting functionality) during runtime.

This dynamic behavior modification capability of the decorators are useful for adapting objects to new functionality without re-writing the original object’s code.

The given design pattern shows the implementation code of the decorated class. In Java, the code for a decorator class would be something like this:



# Design Pattern

## 1. Log.java

```
public abstract class Log {
    public abstract void write(String log);
}
```

## 2. DatabaseLog.java

```
public class DatabaseLog extends Log {
    // Write to the Database
    public void write(String log){
    }
}
```

## 3. DecoratorLog.java

```
public abstract DecoratorLog extends Log {
    protected Log objLog;
    public DecoratorLog(Log objLog) {
        this.objLog = objLog;
    }
    public void write(String log){
        // Write to the Textfile
        objLog.write(log);
    }
}
```

## 4. DecoratorLogA.java

```
public class DecoratorLogA extends
DecoratorLog {
    public DecoratorLogA(Log aLog) {
        super(objLog);
    }
    public void write(String log){
        // Decorator A behavior here
        objLog.write(String log);
    }
}
```

## 5. DecoratorLogB.java

```
public class DecoratorLogB extends
DecoratorLog {
    public DecoratorLogB(Log aLog) {
        super(objLog);
    }
    public void write(String log){
        // Decorator B behavior here
        objLog.write(String log);
    }
}
```

In the given code, the method **write(String log)** of the abstract class **Log** is extended by the decorator class (**DecoratorLog**) and its subclasses to add the new functionality (such as write to the Textfile) to the method.

You can also define an interface in place of an abstract class shown as:

## Log.java

```
interface Log{
    public void write(String log);
}
```

Once you have defined the interface, you can implement its behaviors in any class e.g.

```
public class DatabaseLog implements Log
{public void write(String log){// Decorator A
behavior here}}
```

## II. Facade Design Pattern

The **Facade Pattern** is an object-oriented design pattern, which provides a simplified interface to a larger body of code. This pattern can make the task of accessing a large number of modules much simpler by providing an additional and a higher-level interface layer to an entire subsystem of objects that makes the subsystem easier to use. Thus, it hides the complexities and the implementation of the subsystem from clients.

A **facade** is an object that is placed between the group of objects and its client who accesses the system.

### Facade Pattern : When to use

- When a way of hiding a complex system within a simple interface is required.
- When lots of dependencies between client and its implementation class of an abstraction are required.
- When you want to make subsystems in layers.

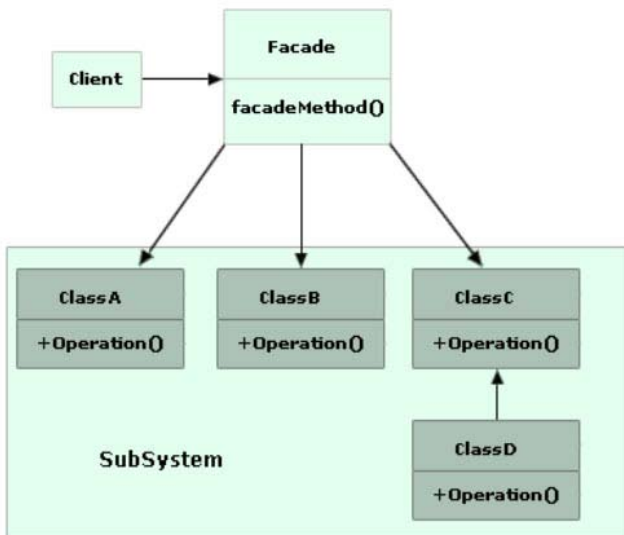
# Design Pattern

## Benefits:

- Provides simple interface to a complex system without reducing options provided by the system.
- Increases the reuse of classes by decoupling the interface from the implementation
- Promotes weak coupling between the subsystem and its clients
- Translates client requests to subsystem that can fulfill request

Decorators and Facade evoke similar images in building architecture, but in design pattern terminology, the Facade is a way of hiding a complex system inside a simpler interface, whereas Decorator adds function by wrapping a class.

**The UML diagram for this pattern is something like this:**



This UML diagram shows that, there can be any number of classes involved in this "Facade" system. There is one client, one facade, and multiple classes underneath the facade. In a typical situation, the facade would have a limited amount of actual code, making calls to lower layers most of the time.

We can understand the facade pattern better using a simple example. Let's consider a **Bank**. This bank has a service provider. In the bank, there are a lot of accounts available e.g. saving A/c, current A/c material, corporate A/c.

You, as a client want to access your different accounts. You just have access to service provider, who knows well about your accounts information to update inquiries regarding your account. Whatever you want, you ask the service provider and he retrieves it out and tells you on showing him the credentials.

**Here, the service provider acts as the facade, as he hides the complexities of the Bank information system.**

**Let us see how the Bank example works.**

## Bank.java

```
public interface Bank {  
    public Inquiries getInquiry(String  
inquiryType); }// End of interface  
}// End of interface
```

The Bank is an interface that only returns Inquiries. The Inquiries are of three types as discussed before:

SavingInquiries, CurrentInquiries and CorporatetInquiries.

All these classes can implement the Inquiries interface.

**Let's have a look at the code for one of the Inquiry.**

## SavingInquiryBank.java

```
public class SavingInquiryBank implements  
Bank { public Inquiries getInquiry ( ) {//  
write functionality of saving  
accountSavingInquiries savinginquiries = new  
SavingInquiries();return savinginquiries;} }//  
End of class
```

Now let's consider the facade ServiceProvider.

# Design Pattern

## ServiceProvider.java

```
public class ServiceProvider {
    /* Implement common method

public Inquiries getInquiry (String inquiryType)
{
public class ServiceProvider {
/* Implement common methodpublic Inquiries
getInquiry (String inquiryType)
{
    if (inquiryType.equals("Saving")) {
        SavingInquiryBank bank = new
        SavingInquiryBank ();SavingInquiries
        savinginquiries =
        (SavingInquiries)bank.getInquiry ();
        return savinginquiries;
    }
    else if (inquiryType.equals("Current")) {
        CurrentInquiryBank bank = new
        CurrentInquiryBank ();
        CurrentInquiries currentinquiries =
        (CurrentInquiries)bank.getInquiry ();
        return currentinquiries;
    }
    else {
        CorporateInquiryBank bank = new
        CorporateInquiryBank ();
        CorporatetInquiries corporatetinquiries =
        (CorporateInquiries)
        bank.getInquiry ();
        return corporatetinquiries;
    }}// End of class
```

This is clear that the complex implementation will be done by ServiceProvider himself. The client will just access the ServiceProvider and ask details of either saving, current or corporate inquiries.

The client program can now create an object of **ServiceProvider** class and call method **getInquiry()** passing as parameter the type of inquiry required. This can be done as follows.

```
new
ServiceProvider().getInquiry("Saving");
```

Here is a simple code of the client program that accesses this façade for getting saving account information.

## Client.java

```
public class Client {
    /*** to get saving inquiries*/
    public static void main(String[] args) {
        ServiceProvider provider = new
        ServiceProvider().getInquiry("Saving");
    } }
// End of class
```

## III. Flyweight Design Pattern

The Flyweight is a software design pattern, useful when there is the need for multiple objects sharing some common information.

In programming languages, some times you may need to generate a very large number of small class instances to represent the entire system. This is usually very memory consuming to keep track of those objects. The Flyweight Pattern reduces this problem and avoids the overhead of large numbers of very similar class-objects.

The Flyweight design pattern provides an approach for handling the classes in a system where these classes are maintained through the extrinsic data that is passed in as arguments.

### When to use Flyweight Pattern:

- The application requires a large number of objects.
- Storage costs are high and difficult to maintain the number of objects.
- The application doesn't depend on object identity.

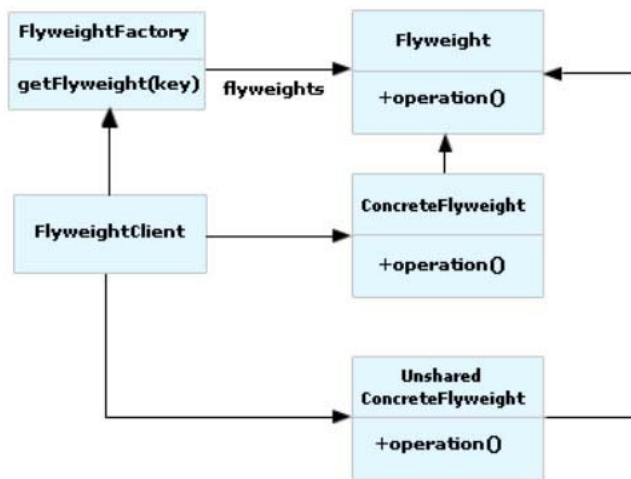
### Benefits:

- Reduction of number of Objects to handle.
- Reduction in memory and on storage devices, if objects are persisted.



# Design Pattern

The UML diagram for this pattern is like this:



This UML diagram shows that, Flyweights are typically instantiated by a **flyweight factory** that creates a limited number of flyweights and sends them out, one at a time to its clients.

Clients don't instantiate flyweights directly; instead they get them from a **Flyweight Factory**. The factory first checks to see if it has a flyweight that fits specific criteria; if so, the factory returns a reference to the flyweight. If the factory can't locate a flyweight for the specified criteria, it instantiates one, adds it to the pool, and returns it to the client.

**Flyweight** declares an interface through which flyweights can receive and act on extrinsic state. **ConcreteFlyweight** implements the **Flyweight** interface and adds storage for intrinsic state, if any in order to share an object. **FlyweightFactory** creates and manages flyweight objects.

We can understand the flyweight pattern better using a simple example. Suppose, you want to show a file system with folders to show the directories or subdirectories, then you don't need to load all the files or directories at one loading time. You may show the upper level folders first. If the user clicks a folder, then load its subdirectories and files. The shared trigger is mouse-clicked. The composite pattern may be combined to define the flyweight system.

Let us see an example of flyweight pattern.

## FlyweightIntr.java

```
interface FlyweightIntr {
    public String getName();
    public String getAddress();
}
```

The **FlyweightIntr** is an interface that returns the name and address of the employees based on their specific criteria "division".

## FlyweightClient.java

```
import java.util.HashMap;
import java.util.StringTokenizer;
import java.util.Vector;

public class FlyweightClient {
    public static void main(String[] args) throws Exception {
        Vector empList = store();
        FlyweightFactory factory =
            FlyweightFactory.getInstance();

        for (int i = 0; i < empList.size(); i++) {
            StringTokenizer st = new StringTokenizer();
            String division = st.nextToken();
            FlyweightIntr flyweight =
                factory.getFlyweight(division);

            // associate the flyweight
            // with the extrinsic data object.

            VCard card = new VCard(name, flyweight);
            card.print();
        }
    }

    private static Vector store() {
        Vector v = new Vector();
        v.add("North");
        v.add("South");
        v.add("North");
        return v;
    }
}
```

# Design Pattern

## FlyweightFactory.java

```
class FlyweightFactory {
private HashMap IstFlyweight;

private static FlyweightFactory factory = new
FlyweightFactory();

private FlyweightFactory() {
IstFlyweight = new HashMap();
}

public synchronized FlyweightIntr
getFlyweight(String divisionName) {
if (IstFlyweight.get(divisionName) == null) {
FlyweightIntr fw = new
Flyweight(divisionName);
IstFlyweight.put(divisionName, fw);
return fw;
} else {
return
(FlyweightIntr)IstFlyweight.get(divisionName);
}
}

public static FlyweightFactory getInstance() {
return factory;
}
```

The client accesses this factory of employees. Every time, the client wants the information, which is accessed through this Factory class. The specifications are passed through the method parameters and new employee information is returned.

## Flyweight.java

```
//Inner flyweight class

private class Flyweight implements
FlyweightIntr {

private String name;
private String addr;

private void setValues(String name, String
addr) {

this.name = name;
this.addr = addr;
}
```

```
private Flyweight(String division) {
if (division.equals("North")) {
setValues("soniya", "addr1");
}
if (division.equals("South")) {
setValues("rahul", "addr2");
}
if (division.equals("East")) {
setValues("Aqil", "addr3");
}
}

public String getName() {
return company;
}

public String getAddress() {
return address;
}
} // end of Flyweight
} // end of FlyweightFactory

class VCard {

String name;
String title;

FlyweightIntr objFW;

public VCard(String n, FlyweightIntr fw) {
name = n;
objFW = fw;
}

public void print() {
System.out.println(name);
System.out.println(name objFW.getAddress());
}
}
```

The other important class is Flyweight. This gets constructed depending on the parameters passed to the method **getName()** and **getAddr()** of class FlyweightFactory.

This class has methods getters and setters for information. In each of the instances of the employees, we can pass the values as method parameters. Hence, we can see that by using the flyweight pattern, we can reduce the instances of the class.

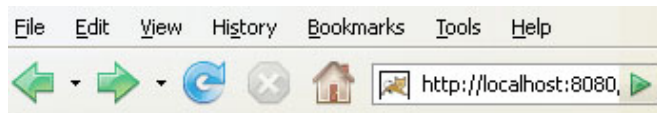


# JSF

```
“Login”  
action="#{LoginBean.CheckValidUser}" />  
</td>  
    </tr>  
    </table>  
</h:form>  
</body>  
</html>  
</f:view>
```

## Description:

JSF tag libraries are included at the top of the file to identify jsf html tags and jsf core tags. JSF outputText tag is used to display text on the page, inputText tag is used to display user editable text field component and inputSecret tag is used to display user editable text field component in which text are displayed in secret form. JSF commandButton tag represents the command button, which have been bounded to “CheckValidUser” method defined in LoginBean managed bean class. According to the result returned by CheckValidUser method and after matching this result with the value defined in navigation rule in configuration file, the next page is displayed.



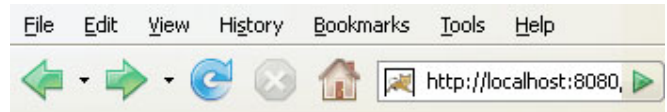
Enter Login ID:

Enter Password:

## 2. loginfail.jsp:

Login Failed. Please try again.

This page is displayed when the user fills incorrect entries in the login page.

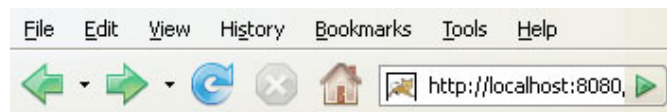


Login Failed. Please try again.

## 3. loginsuccess.jsp

Login Successful.

This page is displayed when the user enters correct entries in the login page.



Login Successful.

## Creating managed bean class LoginBean.java

```
package javajazzup;  
  
public class LoginBean{  
    String loginid;  
    String pwd;  
  
    public LoginBean(){  
  
    public String getLoginid(){  
        return loginid;  
    }  
  
    public void setLoginid(String loginid){  
        this.loginid = loginid;  
    }  
  
    public String getPwd(){  
        return pwd;  
    }  
  
    public void setPwd(String pwd){  
        this.pwd = pwd;  
    }  
  
    public String CheckValidUser(){
```

# JSF

```
if(loginid.equals("JavaJazzUp") &&
pwd.equals("mypwd"))
{
    return "success";
}
else{
    return "fail";
}
}
```

## Description:

In login.jsp file, we have used managed bean's properties and methods. For this LoginBean bean have been created which is nothing but a simple java file which has properties representing username and password, their setter and getter methods and an extra method to check whether the user is genuine. This method returns "success" if user fills username as "JavaJazzUp" and password as "mypwd" and "fail" if user fills any other.

## Working with Configuration file

### 1. faces-config.xml file:

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer
Faces Config 1.1//EN" "http://java.sun.com/
dtd/web-facesconfig_1_1.dtd">
<faces-config>
<managed-bean>
<managed-bean-name>LoginBean
</managed-bean-name>
<managed-bean-class>
javajazzup.LoginBean
</managed-bean-class>
<managed-bean-scope>request
</managed-bean-scope>
</managed-bean>
<navigation-rule>
<from-view-id>/login.jsp</from-view-id>
<navigation-case>
<from-action>
#{
LoginBean.CheckValidUser
}
</from-action>
```

```
<from-outcome>success</from-outcome>
<to-view-id>loginsuccess.jsp</to-view-id>
</navigation-case>
<navigation-case>
<from-action>
#{
LoginBean.CheckValidUser
}
</from-action>
<from-outcome>fail</from-outcome>
<to-view-id>loginfail.jsp</to-view-id>
</navigation-case>
</navigation-rule>
</faces-config>
```

In this file, managed bean "LoginBean" in "javajazzup" package has been registered as LoginBean. This is the name, which will be used in every page to access this bean as we did in login.jsp. Navigation rule has also been defined in this file. If CheckValidUser method returns "success" then "loginsuccess.jsp" page is returned and if "fail" then "loginfail.jsp" is returned to the user.

### 2. web.xml file code:

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

<web-app>
<context-param>
<param-name>
javax.faces.STATE_SAVING_METHOD
</param-name>
<param-value>server</param-value>
</context-param>
<!-- Faces Servlet -->
<servlet>
<servlet-name>Faces Servlet</servlet-name>
<servlet-class>
javax.faces.webapp.FacesServlet
</servlet-class>
<load-on-startup> 1 </load-on-startup>
</servlet>

<!-- Faces Servlet Mapping -->
<servlet-mapping>
```



## JSF

```
<servlet-name>Faces Servlet</servlet-name>  
<url-pattern>*.jsf</url-pattern>  
</servlet-mapping>  
</web-app>
```

<servlet> element maps the "javax.faces.webapp.FacesServlet" servlet class to a symbolic name i.e. Faces Servlet is an alias for "javax.faces.webapp.FacesServlet" servlet . The FacesServlet servlet works as an engine for all JSF applications (handling of all JSF related requests, building component tree of the JSP page, accessing all JSP pages in the application, creating an Event object and passing it to any registered listener). So all requests that need FacesServlet processing must be directed to this servlet. So if we want to invoke this servlet with every request we have to do mapping in <servlet-mapping> element. This is done to map a particular URL pattern with the Faces servlet. According to this file, URL of every request must end with **.jsf**.

### Running the application

Type "http://localhost:8080/JSFLoginApplication/login.jsf" URL in the address bar of your browser and hit enter.



Get  
reliable  
**SERVICES**

- Software Solutions
- E-Commerce Solutions
- Website Development
- Web Promotion
- Web Hosting
- Content Development

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

# AJAX: Redefining Web Applications

By Atul Shinkar

## Ajax: An Introduction

- AJAX stands for **Asynchronous JavaScript And XML**.

- AJAX is not a new programming language, but a new way to use existing standards.

- AJAX is a type of programming made highly popular in 2005 by Google (with Google Suggest).

- With AJAX you can create better, faster, more user-friendly, and interactive web applications (just like Desktop apps).

- AJAX uses **asynchronous** data transfer (HTTP requests) between the browser and the web server, allowing web pages to request small bits of information from the server instead of whole pages, so no page reloads.

- The first use of the term in public was coined by **Jesse James Garrett** of Adaptive Path in February 2005.

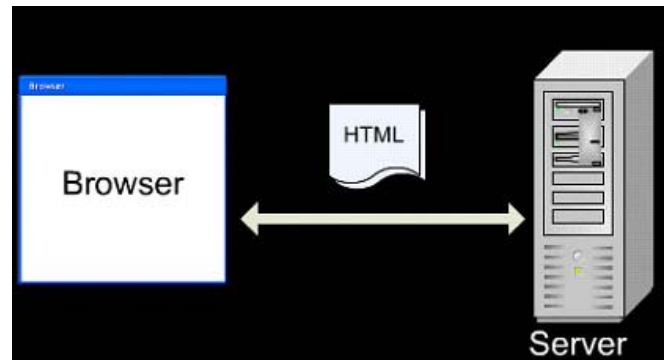
- AJAX incorporates:

- Standards-based presentation using XHTML and CSS
- Dynamic display and interaction using the Document Object Model (DOM)
- Data interchange and manipulation using XML and XSLT
- Asynchronous data retrieval using XMLHttpRequest (JavaScript object)
- JavaScript binding everything together

- Updates the web page data on the fly.

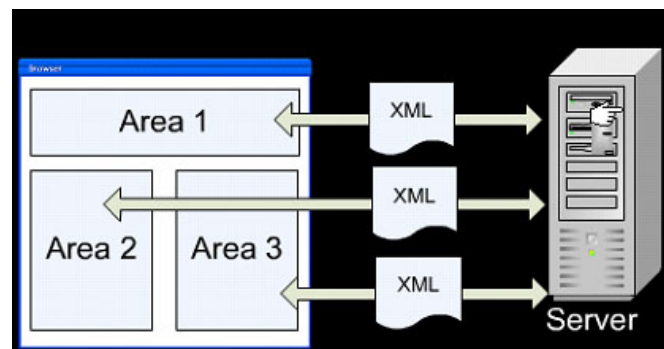
- Uses XMLHttpRequest object to communicate asynchronously with the Server. § All of today's mainstream browsers like Internet Explorer 5.0+, Safari 1.2, Mozilla 1.0 / Firefox, Opera 8+, and Netscape 7 support the XMLHttpRequest object.

## Web Scenario without AJAX:



- One page per request
- One URL for one page
- The whole page updates

## Web Scenario with AJAX:



- Each action has a URL
- Multiple requests per page
- You control what updates

## Detecting the web browser:

- In JavaScript, we use the following code to detect the type of browser, the application is dealing with:

```
var request;  
if (window.XMLHttpRequest) {  
  //non-Microsoft browser  
  request = new XMLHttpRequest( );  
} else if (window.ActiveXObject) {  
  //Microsoft browser  
  request = new  
  ActiveXObject("Microsoft.XMLHTTP");  
}
```

## AJAX Request 'n' Response

## AJAX: Redefining Web Applications

- On some particular event, the JavaScript code sends the request behind the scenes to the server (in Java, the request is sent to the Servlet); the user doesn't even realize that the request is being made, i.e. the request is sent asynchronously, which means that your JavaScript code (and the user) doesn't wait around on the server to respond.
- The server sends data back as a response to your JavaScript code which decides what to do with that data.
- The response data can be simple string (used in case if you're updating only single field on the form) or XML string (used for updating multiple fields on the form).
- The following things happen during AJAX interaction:
  - 1 A client event occurs.
  - 2 An XMLHttpRequest object is created and configured.
  - 3 The XMLHttpRequest object makes a call.
  - 4 The request is processed by the Servlet.
  - 5 The Servlet returns an XML document containing the result.
  - 6 The XMLHttpRequest object calls the *callback()* function and processes the result.
  - 7 The HTML DOM (or UI) is updated.

Code illustrations for AJAX request 'n' response:

### SimpleAJAXDemo.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.01 Transitional//EN">
<html>
<head>
<title>Simple AJAX Demo</title>

<meta http-equiv="keywords"
content="keyword1,keyword2,keyword3">
<meta http-equiv="description"
content="this is my page">
```

```
<meta http-equiv="content-type"
content="text/html; charset=UTF-8">

<script language="javascript">
var req;
function convertToString() {
var num =
document.getElementById("num");
var url = "/ajaxapp/
SimpleAJAXResponse?num=" +
escape(num.value);
if(window.XMLHttpRequest){
req = new XMLHttpRequest();
}
else if(window.ActiveXObject){
req = new
ActiveXObject("Microsoft.XMLHTTP");
}
req.open("Get",url,true);
req.onreadystatechange = callback;
req.send(null);
}
function callback() {
if( req.readyState==4 ){
if( req.status==200 ) {
var strNum =
document.getElementById("numstring");
strNum.value = req.responseText;
}
}
}
}

function clear(){
var num =
document.getElementById("num");
num.value = "";
}
}

function focusIn(){
document.getElementById("num").focus( );
}
}
</script>

</head>

<body onload="focusIn();">
Enter the Number here:
<input type="text" id="num" name="num"
onkeyup="convertToString();">
<br><br>
Equivalent String:
<input type="text" size="20" readonly
```

# AJAX: Redefining Web Applications

```
id="numstring">
</body>
</html>
```

## SimpleAJAXResponseServlet.java

```
import java.io.IOException;
import java.rmi.ServerException;
import javax.servlet.http.*;

public class SimpleAJAXResponseServlet
extends HttpServlet {

public void doGet( HttpServletRequest
request, HttpServletResponse response )
throws ServletException, IOException {

String number =
request.getParameter("num");
String strRepr = null;
if(number != null){
switch(Integer.parseInt(number)){
case 0:
strRepr = "Zero";
break;
case 1:
strRepr = "One";
break;
case 2:
strRepr = "Two";
break;
case 3:
strRepr = "Three";
break;
case 4:
strRepr = "Four";
break;
case 5:
strRepr = "Five";
break;
case 6:
strRepr = "Six";
break;
case 7:
strRepr = "Seven";
break;
case 8:
strRepr = "Eight";
break;
case 9:
strRepr = "Nine";
break;
```

```
default:
strRepr = "Sorry, no conversion
available for the number "+number+";"
}

//Set up the response
response.setContentType("text/xml");
response.setHeader("Cache-Control",
"no-cache");
response.getWriter().write(strRepr);
}
else{
response.setContentType("text/xml");
response.setHeader("Cache-Control",
"no-cache");
response.getWriter().write("?");
}
}
}
```

## Output:

Enter the Number here:

Equivalent String:

When the number 5 is entered, the output is like this

Enter the Number here:

Equivalent String:

When the number 20 is entered, the output is like this

Enter the Number here:

Equivalent String:

In the above example, no xml has been used as we are sending only a single response string, so there is no need to construct any xml. So, in this case, we say that the xml is over killed.

# AJAX: Redefining Web Applications

**Let's construct and parse the response data:**

## I. Constructing and parsing response data using XML:

- There is XML parser is built into most browsers; we just have to leverage the built-in parser.
- On the Servlet side, we can construct XML using the following ways:
  - StringBuffer [common approach]
  - JDOM
  - DOM4J
  - SAX [Simple API for XML]
  - Faster than JDOM & DOM4J
- On the Client side, we've to parse the response XML string using the browser in-built parser.
  - Firefox, Mozilla, Opera, and Safari all use new DOMParser() to get a built-in parser that can parse XML
  - Internet Explorer, on the other hand, uses new ActiveXObject("Microsoft.XMLDOM") to get the Microsoft XML parser.

- Example XML response string:

```
<converted-values>
<decimal>97</decimal>
<hexadecimal>0x61</hexadecimal>
<octal>0141</octal>
<binary>1100001B</binary>
</converted-values>
```

## Code illustrations for Constructing and parsing response data using XML:

### AJAXCharacterDecoderUsingXML.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.01 Transitional//EN">
<html>
<head>
<title>AJAXCharacterDecoder.html</title>
```

```
<meta http-equiv="keywords"
content="keyword1,keyword2,keyword3">
<meta http-equiv="description" content="this
is my page">
<meta http-equiv="content-type"
content="text/html; charset=UTF-8">

<!--<link rel="stylesheet" type="text/css"
href="/styles.css">-->

<script language="javascript">

var request;

function convertToXML( ){

var key = document.getElementById("key");
var keypressed =
document.getElementById("keypressed");
keypressed.value = key.value;
var url = "/ajaxapp/
CharacterDecoderUsingXML?key=" +
escape(key.value);
if(window.XMLHttpRequest){
request = new XMLHttpRequest();
} else{
request = new
ActiveXObject("Microsoft.XMLHTTP");
}
request.open("Get",url,true);
request.onreadystatechange = callback;
request.send(null);
}

function callback( ){
if(request.readyState == 4){
if(request.status == 200){
if(window.XMLHttpRequest){
nonMSPopulate();
} else{
msPopulate();
}
}
}
}

function nonMSPopulate( ){
var response = request.responseText;
var xmlDoc =
document.implementation.createDocument("",null);
var parser = new DOMParser();
var dom =
```





# AJAX: Redefining Web Applications

## CharacterDecoderUsingXMLServlet.java

```
import java.io.IOException;
import java.rmi.ServerException;
import javax.servlet.http.*;

public class CharacterDecoderUsingXMLServlet
extends HttpServlet {

public void doGet( HttpServletRequest
request, HttpServletResponse response )
throws ServletException, IOException {

String strEnteredKey =
request.getParameter("key");
StringBuffer responseXML = null;
System.out.println("\nUsing XML");
System.out.println("Entered key:
"+strEnteredKey);
if( strEnteredKey!="") {
int num = Integer.parseInt(strEnteredKey);
responseXML = new
StringBuffer("\r\n<converted-values>");
responseXML.append
("\r\n<binary>"+Integer.toBinaryString(num)+
"</binary>");
responseXML.append
("\r\n<octal>"+Integer.toString(num,8)+
"</octal>");

responseXML.append("\r\n<decimal>"+num+
"</decimal>");

responseXML.append
("\r\n<hexadecimal>"+Integer.toString(num,16)+
"</hexadecimal>");
responseXML.append("</converted-
values>");

System.out.println("\n"+responseXML.toString());
response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-
cache");
response.getWriter().write
(responseXML.toString());
}
else {
responseXML = new
StringBuffer("\r\n<converted-values>");
responseXML.append("\r\n<binary>?
</binary>");
responseXML.append("\r\n<octal>?
```

```
</octal>");
responseXML.append("\r\n<decimal>?
</decimal>");

responseXML.append("\r\n<hexadecimal>?</
hexadecimal>");
responseXML.append
("</converted-values>");

System.out.println("\n"+responseXML.toString());
response.setContentType("text/xml");
response.setHeader("Cache-Control",
"no-cache");

response.getWriter().write(responseXML.toString());
}
}
}
```

## Output:

### AJAX Character Decoder

Enter the key here:

Key pressed: <input type="text"/>			
Binary	Octal	Decimal	Hexadecimal

When you enter 12, the corresponding out is like as shown:

### AJAX Character Decoder

Enter the key here:

Key pressed: <input type="text" value="12"/>			
Binary	Octal	Decimal	Hexadecimal
1100	14	12	c

**Corresponding output generated at the server console:**

# AJAX: Redefining Web Applications

```
Using XML
Entered key: 12
<converted-values>
  <binary>1100</binary>
  <octal>14</octal>
  <decimal>12</decimal>
  <hexadecimal>c</hexadecimal>
</converted-values>
```

If you do not provide any value, it behaves similar as shown below:

## AJAX Character Decoder

Enter the key here:

Key pressed: <input type="text"/>			
Binary	Octal	Decimal	Hexadecimal
?	?	?	?

When the number 67 is entered, it behaves as:

## AJAX Character Decoder

Enter the key here:

Key pressed: <input type="text" value="67"/>			
Binary	Octal	Decimal	Hexadecimal
1000011	103	67	43

## Corresponding output at the server console:

```
Using XML
Entered key: 67
<converted-values>
  <binary>1000011</binary>
  <octal>103</octal>
  <decimal>67</decimal>
  <hexadecimal>43</hexadecimal>
</converted-values>
```

In the above servlet, we used simple StringBuffer class to create a xml as a response string. This response string is then parsed at the client-side by leveraging the built-in xml parsers in the browser. There are two different

functions given in the html file, i.e. **msPopulate()** for IE browser and **nonMSPopulate()** for non-IE browsers. The extracted data is then placed into the different UI elements of the web page.

## II. Constructing and parsing the response data with JSON:

- One major drawback with XML is speed.
- XML requires two tags per data point, plus extra tags for parent nodes. All this extra data in transmission slows down the data exchange between the client and server.
- It means that the data transmission becomes slower as the response XML string becomes bulkier.
- So, there is another way to send data to the client that is easier to parse and more compact. That alternative is **JSON** [JavaScript Object Notation].

### Advantages of using JSON:

- JSON objects are typically smaller than the equivalent XML documents.
  - Working with them is more memory-efficient.
  - You can parse JSON with JavaScript's **eval()** function – For this you don't need other libraries, and you don't need to worry as much about cross-browser functionality.
- As long as your browser has JavaScript enabled and supports the eval() function, you will be able to interpret the data.
- Following is the data object represented in JSON:

```
{
  "conversion":
  {
    "decimal": "120",
    "hexadecimal": "78",
    "octal": "170",
    "binary": "1111000B"
  }
}
```

# AJAX: Redefining Web Applications

```
}  
}
```

- On Servlet side, we can construct JSON object using
  - StringBuffer [common approach]
  - Using JSON library
  - Can be downloaded from: [http://www.JSON.org/java/json\\_simple.zip](http://www.JSON.org/java/json_simple.zip)

## Code illustrations for Constructing and parsing response data using JSON:

### AJAXCharacterDecoderUsingJSON.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD  
HTML 4.01 Transitional//EN">  
<html>  
  <head>  
    <title>AJAX Character Decoder</title>  
    <meta http-equiv="keywords"  
    content="keyword1,keyword2,keyword3">  
    <meta http-equiv="description"  
    content="this is my page">  
    <meta http-equiv="content-type"  
    content="text/html; charset=UTF-8">  
  
    <!--<link rel="stylesheet" type="text/css"  
    href="/styles.css">-->  
    <script language="javascript">  
      var request;  
      function convertToXML( ){  
  
var key =  
document.getElementById("key");  
var keypressed =  
document.getElementById("keypressed");  
keypressed.value = key.value;  
var url = "/ajaxapp/  
CharacterDecoderUsingJSON?key=" +  
escape(key.value);  
if(window.XMLHttpRequest){  
  request = new XMLHttpRequest();  
} else{  
  request = new  
ActiveXObject("Microsoft.XMLHTTP");  
}  
  request.open("Get",url,true);  
  request.onreadystatechange = callback;  
  request.send(null);  
}  
}
```

```
function callback( ){  
  if(request.readyState == 4){  
    if(request.status == 200){  
      populateUsingJSON( );  
    }  
  }  
}  
  
function populateUsingJSON( ){  
  
  var jsonData = request.responseText;  
  
  var jsonObject = eval('(' +jsonData+')');  
  
  var binary =  
  document.getElementById("binary");  
  binary.value =  
  jsonObject.conversion.binary;  
  var octal =  
  document.getElementById("octal");  
  octal.value = jsonObject.conversion.octal;  
  var decimal =  
  document.getElementById("decimal");  
  decimal.value =  
  jsonObject.conversion.decimal;  
  
  var hexadecimal =  
  document.getElementById("hexadecimal");  
  hexadecimal.value =  
  jsonObject.conversion.hexadecimal;  
  
}  
  
function focusIn( ){  
  document.getElementById("key").focus;  
}  
</script>  
  
</head>  
  
<body onload="focusIn( );">  
<h1>AJAX Character Decoder</h1>  
  
<table>  
<tr>  
<td>  
Enter the key here: &nbsp; &nbsp;  
<input type="text" id="key" name="key"  
  maxlength="2" size="2"  
  onkeyup="convertToXML( );">  
</td>
```

# AJAX: Redefining Web Applications

```
</tr>
</table>
</br>
<table border="1">
<tr>
<td align="center" colspan="5">
Key pressed: &nbsp;
<input type="text" readonly id="keypressed"
maxlength="2" size="2">
</td>
</tr>
<tr>
<td align="center">Binary</td>
<td align="center">Octal</td>
<td align="center">Decimal</td>
<td align="center">Hexadecimal</td>
</tr>
<tr>
<td align="center">
<input type="text" readonly id="binary">
</td>
<td align="center">
<input type="text" readonly id="octal">
</td>
<td align="center">
<input type="text" readonly id="decimal">
</td>
<td align="center">
<input type="text" readonly
id="hexadecimal">
</td>
</tr>
</table>

</body>
</html>
```

## CharacterDecoderUsingJSONServlet.java

```
import java.io.IOException;
import java.rmi.ServerException;
import javax.servlet.http.*;

public class
CharacterDecoderUsingJSONServlet extends
HttpServlet {

    public void doGet( HttpServletRequest
request, HttpServletResponse response
)throws ServletException, IOException {

String strEnteredKey =
```

```
request.getParameter("key");
StringBuffer resp = null;
if( strEnteredKey!="") {
int num = Integer.parseInt(strEnteredKey);
resp = new StringBuffer
("\r\n{\"conversion\":{\"");
resp.append("\r\n\"binary\":
\""+Integer.toBinaryString(num)+"\";");
resp.append("\r\n\"octal\":
\""+Integer.toString(num,8)+"\";");
resp.append("\r\n\"decimal\": \""+num+"\";");
resp.append("\r\n\"hexadecimal\":
\""+Integer.toString(num,16)+"\"}");
response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-
cache");
response.getWriter().write(resp.toString());
}
else{
resp = new
StringBuffer("\r\n{\"conversion\":{\"");
resp.append("\r\n\"binary\": \"?\";");
resp.append("\r\n\"octal\": \"?\";");
resp.append("\r\n\"decimal\": \"?\";");
resp.append("\r\n\"hexadecimal\": \"?\"");
resp.append("\r\n}");
response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-
cache");
response.getWriter().write(resp.toString());
}
}
}
```

## Output:

## AJAX Character Decoder

Enter the key here:

Key pressed: <input type="text"/>			
Binary	Octal	Decimal	Hexadecimal
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>



# AJAX: Redefining Web Applications

When the number 4 is entered, the output looks like:

## AJAX Character Decoder

Enter the key here:

Key pressed: <input type="text" value="4"/>			
Binary	Octal	Decimal	Hexadecimal
100	4	4	4

If you do not provide any value, it behaves similar as shown below:

## AJAX Character Decoder

Enter the key here:

Key pressed: <input type="text"/>			
Binary	Octal	Decimal	Hexadecimal
?	?	?	?

Checking it again with number 27.

## AJAX Character Decoder

Enter the key here:

Key pressed: <input type="text" value="27"/>			
Binary	Octal	Decimal	Hexadecimal
11011	33	27	1b

In the above servlet, we used simple StringBuffer class to create a JSON object as a response string. This response string is then parsed at the client-side by using the JavaScript's **eval()** function.

The extracted data is then placed into the different UI elements of the web page.

## Advantages

- We can build rich Internet applications with AJAX.
- The interface is much more responsive. The user has the feeling that changes are instantaneous.
- Waiting time is reduced.
- If a page section encounters an error,

other sections are not affected (if not logically linked) and the data already entered by the user is not lost.

- Traffic to and from the server is reduced considerably.

## Disadvantages

- The web page cannot connect with the browser history engine.
- AJAX is not meant to be used in every application. One of the main reasons for this stays in the fact that search engines cannot index it. So, keeping this in mind, a much better idea than creating complete AJAX application, it would be better to scatter AJAX features within the application.
- Not all browsers (especially older ones) have complete support for JavaScript or the XMLHttpRequest object.

## Some examples of popular websites using AJAX:

- Following are some popular websites using AJAX:
  - Google Suggest
  - Google Maps
  - GMail
  - Google Reader
  - Blogger
  - Flickr
  - YouTube

Ajax isn't a technology: it's a group of ideas that, used together, have proven very powerful to make the web applications more interactive, but is not meant to be used in every web application.

# Tips & Tricks

## 1. Splash Screen in Java 6.0

Splash screens are standard part of many GUI applications to let the user know about starting of the application. AWT/Swing can be used to create splash screens in Java. Prior to Java SE 6, you need to create a window and include an image in it when main method starts to get the behavior of splash screen. But to get the output, Java run time needs to be fully initialized before the window appeared. A new feature in Java SE 6 allows an application to show a splash screen even before the Java runtime starts with the help of a new command-line option that enables the image to be displayed more quickly to the user i.e. even before starting of Java runtime. If you are using command line to run the application, generate splash screen using **-splash** command line switch followed by a colon, which is followed by the image name. For example, to display button.png image file as splash screen when running SplashScreenExample.java file, you can type the following in command line.

### **java -splash:button.png SplashScreenExample**

The splash screen image can be of GIF, PNG, or JPEG format and support transparency, translucency, and animation. When the application creates its first window the splash screen disappears.

### **Example Code:**

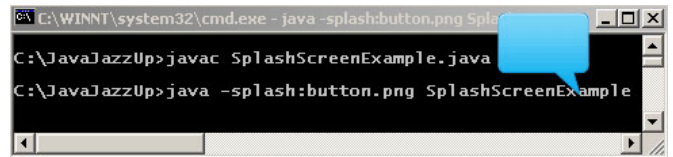
```
import javax.swing.*;
import java.awt.*;


public class SplashScreenExample {
    public static void main(String args[]) {
        Runnable r = new Runnable() {
            public void run() {
                try {
                    Thread.sleep(1500);
                }
                catch (InterruptedException e) {
                    System.out.println(e.getMessage());
                }
            }
        };
        JFrame frame = new JFrame("Splash Screen Example!");
        frame.setDefaultCloseOperation
```

```
(JFrame.EXIT_ON_CLOSE);
JLabel label = new JLabel("Welcome to
JavaJazzUp", JLabel.CENTER);
label.setBackground(Color.RED);
frame.add(label, BorderLayout.CENTER);
frame.setSize(200, 100);
frame.setVisible(true);
    }
};
EventQueue.invokeLater(r);
}
}
```

### **Compile javac SplashScreenExample.java**

### **Run: java -splash:button.png SplashScreenExample**



The image  can be seen in the figure above. It disappears before frame gets visible. The frame can be seen in the figure below.



## 2. Creating Tabs with swing

Many applications need to keep group of components and switch between them. For this tab componets are useful in which we can group components and switch between them clicking those tabs. For this, JtabbedPane class in

## Tips & Tricks

javax.swing package can be used in Java. To create a tabbed pane, instantiate JTabbedPane, create components you wish it to display, and then add components to the tabbed pane using addTab() method.

### Example Code:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JTabbedPaneExample implements
ActionListener {
    JFrame frame;
    JTabbedPane tabPane;
    JButton addTab;
    ImageIcon close;
    Dimension size;
    int tabCounter = 0;

    public static void main(String[] args) {
        JTabbedPaneExample jtab = new
JTabbedPaneExample();
    }

    public JTabbedPaneExample() {
        // Create a frame
        frame = new JFrame();
        // Create the tabbed pane.
        tabPane = new JTabbedPane();
        // Create a button to add a tab
        addTab = new JButton("Add Tab");
        addTab.addActionListener(this);
        // Create an image icon to use as a
close button
        close = new ImageIcon("C:/
JAVAJAZZUP/tabClose.gif");
        size = new
Dimension(close.getIconWidth()+1,
close.getIconHeight()+1);
        //Adding into frame
        frame.add(tabPane,
BorderLayout.CENTER);
        frame.add(addTab,
BorderLayout.NORTH);
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation
(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

```
public void actionPerformed(ActionEvent e)
{
    JLabel label = null;
    final JPanel panel = new JPanel();
    // Create a panel to represent the tab
    JPanel tab = new JPanel();
    tab.setOpaque(false);
    try{
        String str =
JOptionPane.showInputDialog(null, "Enter Tab
Name : ", "JavaJazzUp", 1);
        if (str.length() == 0){
            JOptionPane.showMessageDialog(null,
"Please Enter the Tab Name : ", "JavaJazzUp",
1);
        }
        else if (str != null){
            label = new JLabel(str);
        }
        else{
            JOptionPane.showMessageDialog(null,
"You pressed cancel button.", "JavaJazzUp",
1);
        }
        JButton tabClose = new JButton(close);
        tabClose.setPreferredSize(size);
        tabClose.addActionListener(new
ActionListener() {
            public void
actionPerformed(ActionEvent e) {
                int tNum =
tabPane.indexOfComponent(panel);
                tabPane.removeTabAt(tNum);
            }
        });
        tab.add(label, BorderLayout.WEST);
        tab.add(tabClose, BorderLayout.EAST);
        tabPane.addTab(null, panel);
        tabPane.setTabComponentAt(
tabPane.getTabCount()-1, tab);
    }
    catch(Exception ex){}
}
```

## Tips & Tricks

}

**Compile and run this program:**

```
C:\WINNT\system32\cmd.exe - java JTabbedPaneExample
C:\JavaJazzUp> javac JTabbedPaneExample.java
C:\JavaJazzUp> java JTabbedPaneExample
```

**Output:**

After running this program, you get:



If you click "Add Tab" button then the input box appears on the screen:



Suppose, you fill "Java" in the input box and click the "OK" button:



After clicking the "OK" button, you get a tab named "Java".



you don't give any name to the input box then it gives:



In the same way, you can add many tabs. For example, here is one more tab named "JavaJazzUp" in the figure below:



### 3. Sending jar file as a Servlet Response

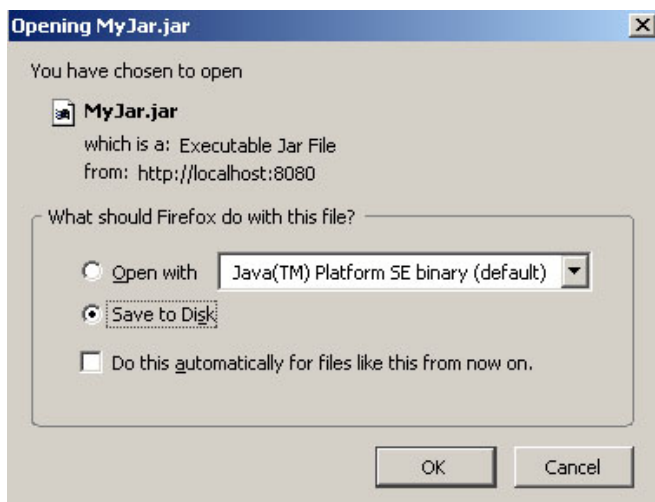
Sometimes, the user needs to download any jar file. This is the example where the "SendJar" servlet is responsible to send the jar file to the user when requested. This code is given below. The important part of this program is to set the content type, which intimates the browser about the type of content to be sent by the servlet. An HTTP response header named

## Tips & Tricks

content-disposition allows the servlet to specify information about the file's presentation. This is used to indicate that the content should be opened separately not in the browser; it should not be displayed automatically, suggesting the file name. In this example, the file will be downloaded with the name "MyJar.jar".

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SendJar extends HttpServlet{
public void doGet(HttpServletRequest req,
HttpServletResponse res) throws
ServletException, IOException{
res.setContentType("application/jar");
res.addHeader("Content-Disposition",
"attachment; filename=MyJar.jar");
ServletContext sc = getServletContext();
InputStream is = sc.getResourceAsStream("/
MyJar.jar");
int len = 0;
byte[] buffer = new byte[1024];
OutputStream os = res.getOutputStream();
while((len = is.read(buffer)) != -1){
os.write(buffer,0,len);
}
os.flush();
os.close();
}
}
```



### 4. Chat Server

Chat server is a standalone application that is

the combination of two-application, server application (which runs on server side) and client application (which runs on client side). This application is used for chatting in LAN. You must be connected with the server before chatting and then your message will be broadcast to every client.

This application uses some core java features like swing, collections, networking, I/O streams and threading also. In this application we have one server and number of clients (which are to be communicated with each other). To make a server we have to run the MyServer.java file at any system on the network that we want to make server and for client we have to run MyClient.java file on the system that we want to make client. To run the whole client operation we can run the Login file (Login.java). It can directly call client file (MyClient.java).

### Client Side application

Before programming the functionality of client side application we need to take the identification of the user, for example user name so that it can be used further in the next client application. So in this application, "Login.java" creates the login frame that consists of one textfield and the login button. After hitting the login button it shows the next frame that is Client Frame that consists of one textfield to write the message, one send button to send it and two list boxes, one is to show all the messages and the other to show all the user names. This frame has one more button that is Logout button for terminating the chat.

### Login.java

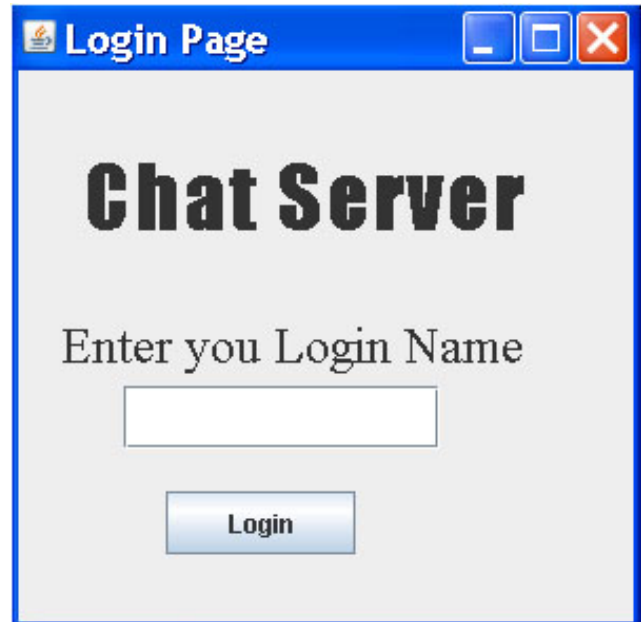
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

// Login class which takes a user name and
// passes it to client class
public class Login implements ActionListener{
JFrame frame1;
JTextField tf;
JButton button;
JLabel heading;
JLabel label;
```



## Tips & Tricks

```
public static void main(String[] args){
    new Login();
}
public Login(){
    frame1 = new JFrame("Login Page");
    tf=new JTextField();
    button=new JButton("Login");
    heading=new JLabel("Chat Server");
    heading.setFont(new Font("Impact",
Font.BOLD,40));
    label=new JLabel("Enter you Login Name");
    label.setFont(new Font("Serif", Font.PLAIN,
24));
    JPanel panel = new JPanel();
    button.addActionListener(this);
    panel.add(heading);panel.add(tf);panel.add(label);
    panel.add(button);
    heading.setBounds(30,20,280,80);
    label.setBounds(20,100,250,60);
    tf.setBounds(50,150,150,30);
    button.setBounds(70,200,90,30);
    frame1.add(panel);
    panel.setLayout(null);
    frame1.setSize(300, 300);
    frame1.setVisible(true);
    frame1.setDefaultCloseOperation
(JFrame.EXIT_ON_CLOSE);
}
// pass the user name to MyClient class
public void actionPerformed(ActionEvent e){
    String name="";
    try{
        name=tf.getText();
        frame1.dispose();
        MyClient mc=new MyClient(name);
    }catch (IOException te){}
}
}
```



### MyClient.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.Iterator;

public class MyClient extends WindowAdapter
implements ActionListener{
    JFrame frame;
    JList list;
    JList list1;
    JTextField tf;
    DefaultListModel model;
    DefaultListModel model1;
    JButton button;
    JButton lout;
    JScrollPane scrollpane;
    JScrollPane scrollpane1;
    JLabel label;
    Socket s,s1,s2;
    DataInputStream din;
    DataOutputStream dout;
    DataOutputStream dlout;
    DataOutputStream dout1;
    DataInputStream din1;
    String name;
```

## Tips & Tricks

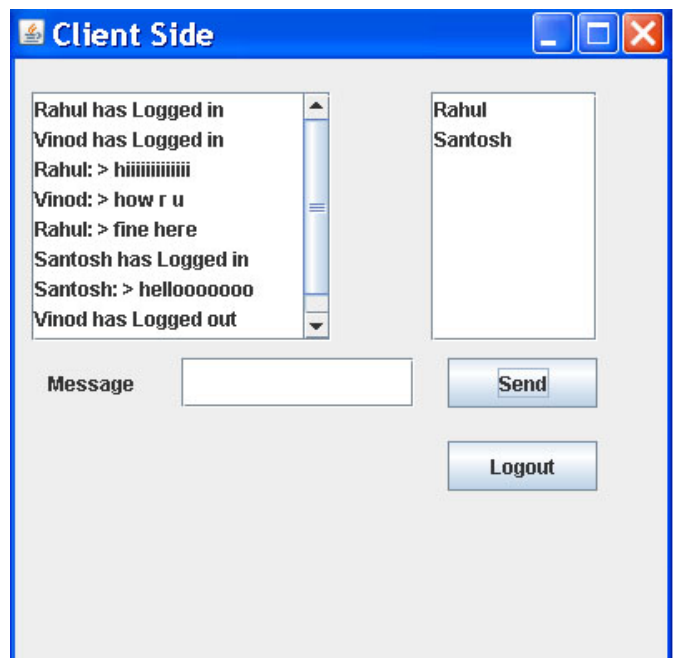
```
MyClient(String name)throws IOException{
    frame = new JFrame("Client Side");
    tf=new JTextField();
    model=new DefaultListModel();
    model1=new DefaultListModel();
    label=new JLabel("Message");
    list=new JList(model);
    list1=new JList(model1);
    button=new JButton("Send");
    lout=new JButton("Logout");
    scrollpane=new JScrollPane(list);
    scrollpane1=new JScrollPane(list1);
    JPanel panel = new JPanel();
    button.addActionListener(this);
    lout.addActionListener(this);
    panel.add(tf);panel.add(button);
    panel.add(scrollpane);
    panel.add(label);panel.add(lout);
    panel.add(scrollpane1);
    scrollpane.setBounds(10,20,180,150);
    scrollpane1.setBounds(250,20,100,150);
    label.setBounds(20,180,80,30);
    tf.setBounds(100,180,140,30);
    button.setBounds(260,180,90,30);
    lout.setBounds(260,230,90,30);
    frame.add(panel);
    panel.setLayout(null);
    frame.setSize(400, 400);
    frame.setVisible(true);
    frame.setDefaultClose
    Operation(JFrame.EXIT_ON_CLOSE);
    this.name=name;
    frame.addWindowListener(this);
    s=new Socket("localhost",1004);
    //creates a socket object
    s1=new Socket("localhost",1004);
    s2=new Socket("localhost",1004);
    //create inputstream for a particular
    socket
    din=new
    DataInputStream(s.getInputStream());
    //create outputstream
    dout=new
    DataOutputStream(s.getOutputStream());
    //sending a message for login
    dout.writeUTF(name+" has Logged in");
    dlout=new
    DataOutputStream(s1.getOutputStream());
    dout1=new
    DataOutputStream(s2.getOutputStream());
    din1=new
```

```
DataInputStream(s2.getInputStream());
// creating a thread for mantaning the list of
user name
    My1 m1=new
My1(dout1,model1,name,din1);
    Thread t1=new Thread(m1);
    t1.start();
//creating a thread for receiving message
    My m=new My(din,model);
    Thread t=new Thread(m);
    t.start();
}
public void actionPerformed(ActionEvent e){
    // sending message
    if(e.getSource()==button){
        String str="";
        str=tf.getText();
        tf.setText("");
        str=name+": > "+str;
        try{
            dout.writeUTF(str);
            System.out.println(str);
            dout.flush();
        }catch(IOException
ae){System.out.println(ae);}
    }
    // client logout
    if (e.getSource()==lout){
        frame.dispose();
        try{
            //sending the message for logout
            dout.writeUTF(name+" has Logged out");
            dlout.writeUTF(name);
            dlout.flush();
            Thread.currentThread().sleep(1000);
            System.exit(1);
        }catch(Exception oe){}
    }
}
public void windowClosing(WindowEvent w){
    try{
        dlout.writeUTF(name);
        dlout.flush();
        Thread.currentThread().sleep(1000);
        System.exit(1);
    }catch(Exception oe){}
}
}
// class is used to mantaning the list of user
name
```

## Tips & Tricks

```
class My1 implements Runnable{
    DataOutputStream dout1;
    DefaultListModel model1;
    DataInputStream din1;
    String name,lname;
    ArrayList alname=new ArrayList();
    ObjectInputStream obj;
    int i=0;
    My1(DataOutputStream
dout1,DefaultListModel model1,
        String name,DataInputStream din1){
        this.dout1=dout1;
        this.model1=model1;
        this.name=name;
        this.din1=din1;
    }
    public void run(){
        try{
            // write the user name in output
stream
            dout1.writeUTF(name);
            while(true){
                obj=new ObjectInputStream(din1);
                //read the list of user names
                alname=(ArrayList)obj.readObject();
                if(i>0)
                    model1.clear();
                Iterator i1=alname.iterator();
                System.out.println(alname);
                while(i1.hasNext()){
                    lname=(String)i1.next();
                    i++;
                    //add the user names in list box
                    model1.addElement(lname);
                }
            }
        }catch(Exception oe){}
    }
}
```

```
}
//class is used to receive message
class My implements Runnable{
    DataInputStream din;
    DefaultListModel model;
    My(DataInputStream din, DefaultListModel
model){
        this.din=din;
        this.model=model;
    }
    public void run(){
        String str1="";
        while(true){
            try{
                str1=din.readUTF(); // receive the
message
                // add the message in list box
                model.addElement(str1);
            }catch(Exception e){}
        }
    }
}
```



### Server side application

Server side application is used to get the message from any client and broadcast to each and every client. And this application is also used to maintain the list of users and broadcast this list to everyone.

## Tips & Tricks

### The Server side application follows these steps

1. Create a server socket by which a server can accept connections from clients across a network.

```
ServerSocket ss = new ServerSocket(1004);
```

2. Use accept() method of ServerSocket to connect to a client. It returns Socket object. Add this client socket into arraylist.

```
Socket s = ss.accept();
ArrayList al2 = new ArrayList();
Al2.add(s);
```

3. After getting the client socket it creates a thread and make DataInputStream for this socket. Then it reads user name and add it to arraylist and this arraylist object is written in ObjectOutputStream of each client by using iterator.

```
DataInputStream din1=new
DataInputStream(s.getInputStream)
ArrayList alname=new ArrayList();
alname.add(din1.readUTF());
Iterator i1=al2.iterator();
Socket st1;
while(i1.hasNext()){
st1=(Socket)i1.next();
dout1=new
DataOutputStream(st1.getOutputStream());
ObjectOutputStream obj=new
ObjectOutputStream(dout1);
obj.writeObject(alname);
}
```

4. After this it makes a new thread and makes one DataInputStream for reading the messages which is sent by the client and after reading the message it creates the DataOutputStream for each socket and writes this message in each client output stream through iterator.

```
String str=din.readUTF();
Iterator i=al.iterator();
Socket st;
```

```
while(i.hasNext()){
st=(Socket)i.next();
dout=new
DataOutputStream(st.getOutputStream());
dout.writeUTF(str);
dout.flush();
}
```

5. If any client logs out then server receives the client name. Server removes it from the arraylist and sends this updated arraylist to all clients.

```
sname=ddin.readUTF();
alname.remove(sname);
```

### MyServer.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class MyServer{
    ServerSocket ss;
    Socket s;
    ArrayList al=new ArrayList();
    ArrayList al1=new ArrayList();
    ArrayList al2=new ArrayList();
    ArrayList alname=new ArrayList();
    Socket s1,s2;
    MyServer()throws IOException{
        ss=new ServerSocket(1004); // create
server socket
        while(true){
            s=ss.accept(); //accept the client socket
            s1=ss.accept();
            s2=ss.accept();
            al.add(s); // add the client socket in
arraylist
            al1.add(s1);
            al2.add(s2);
            System.out.println("Client is Connected");
//new thread to maintain the list of user name
            MyThread2 m=new
MyThread2(s2,al2,alname);
            Thread t2=new Thread(m);
            t2.start();
//new thread to receive and send message
            MyThread r=new MyThread(s,al);
            Thread t=new Thread(r);
            t.start();
```

## Tips & Tricks

```
// new thread to update the list of user name
    MyThread1 my=new
MyThread1(s1,al1,s,s2);
    Thread t1=new Thread(my);
    t1.start();
}
}
public static void main(String[] args){
    try{
        new MyServer();
    }catch (IOException e){}
}
}
//class is used to update the list of user name
class MyThread1 implements Runnable{
    Socket s1,s,s2;
    static ArrayList al1;
    DataInputStream ddin;
    String sname;
    MyThread1(Socket s1,ArrayList al1,Socket
s,Socket s2){
    this.s1=s1;
    this.al1=al1;
    this.s=s;
    this.s2=s2;
}
public void run(){
    try{
        ddin=new
DataInputStream(s1.getInputStream());
        while(true){
            sname=ddin.readUTF();
            System.out.println("Exit :"+sname);
//remove the logged out user name from
arraylist
            MyThread2.alname.remove(sname);
            MyThread2.every();
            al1.remove(s1);
            MyThread.al.remove(s);
            MyThread2.al2.remove(s2);
            if(al1.isEmpty())
                System.exit(0); //all clients are logged out.
        }
    }catch(Exception ie){}
}
}
// class is used to maintain the list of all online
users
class MyThread2 implements Runnable{
    Socket s2;
    static ArrayList al2;
```

```
static ArrayList alname;
static DataInputStream din1;
static DataOutputStream dout1;

    MyThread2(Socket s2,ArrayList al2,ArrayList
alname){
        this.s2=s2;
        this.al2=al2;
        this.alname=alname;
    }
    public void run(){
        try{
            din1= new
DataInputStream(s2.getInputStream());
// store the user name in arraylist
            alname.add(din1.readUTF());
            every();
        }catch(Exception oe){}
    }
// send the list of user name to all clients
static void every()throws Exception{
    Iterator i1=al2.iterator();
    Socket st1;
    while(i1.hasNext()){
        st1=(Socket)i1.next();
        dout1=new
DataOutputStream(st1.getOutputStream());
        ObjectOutputStream obj=new
ObjectOutputStream(dout1);
//write the list of users in stream of all clients
        obj.writeObject(alname);
        dout1.flush();
        obj.flush();
    } }
//class is used to receive the message and
//send it to all clients
class MyThread implements Runnable{
    Socket s;
    static ArrayList al;
    DataInputStream din;
    DataOutputStream dout;
```



## Tips & Tricks

```
MyThread(Socket s, ArrayList al){
    this.s=s;
    this.al=al;
}
public void run(){
    String str;
    int i=1;
    try{
        din=new
DataInputStream(s.getInputStream());
    }catch(Exception e){}

    while(i==1){
        try{
            str=din.readUTF(); //read the message
            distribute(str);
        }catch (IOException e){}
    }
}
// send it to all clients
public void distribute(String str)throws
IOException{
    Iterator i=al.iterator();
    Socket st;
    while(i.hasNext()){
        st=(Socket)i.next();
        dout=new
DataOutputStream(st.getOutputStream());
        dout.writeUTF(str);
        dout.flush();
    }
}
}
```

**Get  
More and More  
Tutorials**

**FIND QUALITY**

**STUFF ON**

**JAVA TECHNOLOGY**

**AT**



<http://www.roseindia.net>

# 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](mailto:deepak@roseindia.net)



**Coffee with  
JavaFX**

**Enjoy  
A  
New  
Flavour**

**Visit us  
<http://www.roseindia.net>**

**All Cool Jobs**



**Shape Your Career with allcooljobs**

All Cool jobs is a Jobs Job searching site. Just Log on to [www.allcolljobs.com](http://www.allcolljobs.com) and get a free registration.

# Original Perception



## **CBI's failure to act against influential**

For some time judiciary has pronounced such welcoming judgements which have been instrumental in enhancing the faith of people in the criminal judicial system of the country. But there is one agency which has often failed to deliver its honest services on this front. The central investigating agency CBI has been chided by the court in many cases for not dealing with the case seriously especially when the accused ones are from the influential background mainly from political and administrative.