JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
JUIT
विद्या तत्व ज्योतिसम:

Jaypee University of Information Technology
Solan (H.P.)
LEARNING RESOURCE CENTER

Acc. Num. SP03020 Call Num:

General Guidelines:

- Library books should be used with great care.
- Tearing, folding, cutting of library books or making any marks on them is not permitted and shall lead to disciplinary action.
- Any defect noticed at the time of borrowing books must be brought to the library staff immediately. Otherwise the borrower may be required to replace the book by a new copy.
- The loss of LRC book(s) must be immediately brought to the notice of the Librarian in writing.

Learning Resource Centre-JUIT
SP03020

JAVA CHAT SERVER

Submitted in partial fulfillment of the
requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

By:

ARPIT ROCHWANI (031259)
AJOYDEEP SINGH (031266)
RAHUL SHARMA (031278)
AKSHUNN SINGHAL (031416)

Learning Resource Centre
JUIT
Waknaghat Solan (H.P.)

**DEPARTMENT OF COMPUTER SCIENCE
ENGINEERING AND INFORMATION TECHNOLOGY
JAYPEE UNIVERSITY OF INFORMATION
TECHNOLOGY-WAKNAGHAT**

CERTIFICATE

This is to certify that the work entitled, "Java Chat Server" submitted by "Arpit Rochwani (031259) , Ajoydeep Singh (031266), Rahul Sharma (031278), and Akshunn Singhal (031416)" in partial fulfillment for the award of degree of Bachelor Of Technology in Computer Science Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Project Supervisor:

Schandra
26.05.07

Mr. Satish Chandra

Senior Lecturer
Department of Computer Science Engineering and Information Technology,
Jaypee University of Information Technology,
Waknaghat, Solan – 173215, Himachal Pradesh,
INDIA.

ABSTRACT

We intend to build chatting software using JAVA (swings) in which we start a central Server which can be accessed by any Client. Each user can run the client program and connect to server to start chatting. Whenever a User logs in a Tray Icon is created for each User. All clients and server will have list of online users. List is updated as soon as the status of some client changes. There is one main chat room in which all messages can be seen by all clients. Users can also choose to chat in private with any one on the list using Conferencing option. Users are prohibited to use certain abusive words. The Users using such words will be warned during the chat and they won't be displayed. File transfer between users is also enabled .

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

CHAPTER 2

NETWORKING BASICS

CHAPTER 3

MODELLING DESIGN AND FUNCTIONS USED

CHAPTER 4

FUTURE SCOPES AND TESTING

LIST OF FIGURES AND TABLES

CHAPTER 1
INTRODUCTION

The fundamental networking capabilities are defined by classes and interfaces of package *java.net*, through which Java offers *socket-based communication* that enable applications to view networking as streams of data. The classes and interfaces of package java.net also offer *packet - based communications* that enable individual packets of information to be transmitted-this is commonly used to transmit audio and video over the Internet.

Our discussion of networking focuses on both sides of a *client – server relationship*. The client requests that some action be performed, and the *server* performs the action and responds to the client. A common implementation of the request - response model is between World Wide Web browsers and World Wide Web servers.

When a user selects a Web site to browse through a browser (the client application), request is sent to the appropriate Web server (the server application). The server normally responds to the client by sending an appropriate HTML Web page.

1.1.PROJECT SPECIFICATIONS

1.1.1. Hardware Specification:

Operating System-Microsoft Windows XP professional 2002, SP2

Primary Memory- 256 MB RAM

Processor-Intel® Pentium® 4 CPU 2.28 GHz

Secondary Memory-80 HDD

1.1.2. Software Specification

Front End- Java (jdk-6-windows-i586.exe).

1.1.3. Setup for working on Java Platform

Step 1: Run the jdk-6-windows-i586 setup file.

Step 2: Open the console Window.

Step 3: Reach the specified path where all the java files exist (C: /java/bin).

Step 4: Compile all java files using javac <filename>.java .

Step 5: run the files using java <filename>.

Step 6: Run the files using java <filename>.

Step 7: Result is obtained.

1.2. JAVA

Java has gained enormous popularity since it first appeared. Its rapid rise and wide acceptance is due to its design and programming features, particularly because you can write a program once, and run it anywhere. Java was chosen as the programming language for network computers (NC). The popularity and usage of Java is well explained by a statement in Java language white paper by Sun Microsystems that: "Java is simple, object-oriented, interpreted, secure, portable, and dynamic."

(i) Simplicity

Java is a much simpler and easy to use programming language when compared to the other object-oriented programming languages as it easy to make interfaces and does not make use of pointers.

Java is Object-oriented programming and models the real world. Everything in the world can be modeled as an object. Java is centered on creating objects, manipulating objects, and making objects work together.

(ii) Portability: (Platform Independence)

One of the most compelling reasons for Java's popularity is its platform independence. Java runs on most major hardware and software platforms, including Windows 95 and NT, the Macintosh, and several varieties of UNIX. All Java-compatible browsers support Java applets. By moving existing software to Java, you are able to make it instantly compatible with these software platforms. JAVA programs become more portable. Any hardware and operating system dependencies are removed.

(iii) Java is transcribed. (Uses bytecodes)

This means that an interpreter is needed in order to run Java programs. The programs are compiled into a Java Virtual Machine code called bytecode. The bytecode is machine independent and is able to run on any machine that has a Java interpreter. Normally, a compiler will translate a high-level language program to machine code and the code is able to only run on the native machine. If the program is run on other machines, the program has to be recompiled on the native machine. With Java, the program need only be compiled once, and the bytecode generated by the Java compiler can run on any platform.

(iv) The Virtual Machine: Java VM

This VM acts between the Java program and the machine it is running on, brings in the concept of the "abstract computer" that executes the Java code and guarantees certain behaviors regardless of the underlying hardware or software platform. Java compilers turn Java programs into platform-neutral "byte code" that the machine-specific VM interprets along with converting into assembly language for a particular machine.

The Java VM also enforces security policies. It provides a sandbox which defines the limits about what java programmes can do. A Java applet cannot, for example, peek into arbitrary files on the machine it's running on. The most recent version of Java from Sun, known as Java Development Kit (JDK) 1.6, though, provides no consistent method for an applet to request restricted system resources.

(v) Multimedia: Images, Sounds and Animation

The "icing on the cake" for Java is MULTIMEDIA - Sounds, Images, Graphics and Video. In this growing age of multimedia, every task demands good Multimedia applications, sound or graphic technology capabilities. Multimedia demands incredible computing power and only recently, computers supporting it are becoming widespread.

Among the image formats supported by Java is the Graphics Interchange Format .GIF and Joint Photography Experts Group .JPEG. Among the audio formats are AIFF, AU and WAV.

(vi) Security

Java considers security as a part of its design. The Java language, compiler, interpreter, and runtime environment were each developed with security in mind. The compiler, interpreter, and Java-compatible browsers all contain several levels of security measures that are designed to reduce the risk of security compromise, loss of data and program integrity, and damage to system users.

(vii) Reliability

Security and reliability are inter related. Security measures cannot be implemented with any degree of assurance without a reliable framework for program execution. Java provides multiple levels of reliability measures, beginning with the Java language itself. The Java compiler provides several levels of additional checks to identify any mismatches and other inconsistencies. The Java runtime system duplicates many of the checks performed by the compiler and performs additional checks to verify that the executable bytecode form a valid Java program.

(viii) Java is Portable and dynamic

One advantage of Java is that its programs can run on any platform without having to be recompiled. This is one positive aspect of portability. It goes on even further to ensure that there are no platform-specific features on the Java language specification. Having a fixed size for numbers makes Java programs portable. For example in Java, the size of the integer is the same on every platform, as is the behavior of arithmetic.. The Java environment itself is portable to new hardware and operating systems, and in fact, the Java compiler itself is written in Java.

The Java programming language was designed to adapt to an evolving environment. New methods and properties can be added freely in a class without affecting their clients. Also, Java is able to load classes as needed at runtime. As an example, you have a class called 'Square'. This class has a property to indicate the color of the square, and a method to calculate the area of the square. You can add a new property to the 'Square' class to indicate the length and width of the square, and a new method to calculate the perimeter of the square, and the original client program that uses the 'Square' class remains the same.

1.3 JAVA SWING

We have tentatively chosen Swing for implementing our user interface components since it meets many of our needs. First of all, it reportedly runs well on all of the platforms we want to support, namely Windows, Linux. Second, Swing is very well-documented, well-tested, and has the largest developer base among the UI toolkits we investigated. Third, Swing is implemented in Java, a language that our team is comfortable using.

Swing is a GUI toolkit for Java. It is one part of the Java Foundation Classes (JFC). Swing includes graphical user interface (GUI) widgets such as text boxes, buttons, split-panes, and tables.

Swing widgets provide more sophisticated GUI components than the earlier Abstract Window Toolkit. Since they are written in pure Java, they run the same on all platforms, unlike the AWT which is tied to the underlying platform's windowing system. Swing supports pluggable look and feel – not by using the native platform's facilities, but by roughly emulating them. This means you can get any supported look and feel on any platform. The disadvantage of lightweight components is possibly slower execution. The advantage is uniform behavior on all platforms.

Swing's most important advantage is the cross-platform support, which enables the developers to build applications that run on Windows, Mac and Linux. In addition, Swing provides a very rich set of components and features that can easily meet the requirements of many types of applications, such as administration consoles, development tools and business applications.

Swing can be used to create user interfaces that are very intuitive and easy-to-use, implementing all the features that a user would expect in a modern desktop application, such as multi-document interfaces, professional-looking menus, toolbars, contextual popup menus, drag-and-drop, and so on.

It is easy to understand how Swing works internally. Knowing what happens under the hood allows the developers to efficiently use the Swing framework and maximize the application's performance.

Swing application is easy to maintain. The code, the developers produce is well documented and it's easy to modify it when new user requirements must be implemented or when existing business logic must be changed. The Swing developers use best practices and perform extensive testing to guarantee a very good quality of the code.

1.4 FEATURES OF JAVA SWING

A Visual Guide to Swing Components (Java Look and Feel)

Swing is primarily known for its rich set of GUI components. A menu of Swing's components, grouped by type, can be used to create the Java look and feel.

A Visual Guide to Swing Components (Windows Look and Feel)

A menu of Swing's components can also create the Windows look and feel.

Pluggable Look and Feel

This architecture allows a program to have control over its appearance.

Data Transfer

Most programs will want to use drag and drop or cut, copy and paste.

Internationalization and Localization

Internationalizing an application makes it easy to tailor it to the customs and languages of end users around the world.

Accessibility

Making your program accessible means that it can be used, without modification, by anyone with permanent or temporary disabilities who may require special devices. And, in many countries, making programs accessible is the law.

Integrating with the Desktop

An application that is well integrated with the desktop will, where appropriate, allow the user to launch the default mail application or internet browser, pre-populating text fields as needed. It will also allow the user to launch another application to open, edit or print a file associated with that application.

1.5 JAVA (LOOK AND FEEL)

UI- Button

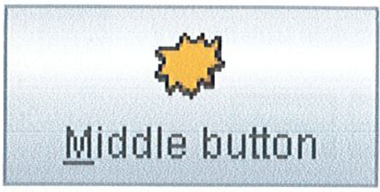


fig 1.1

UI- Text Field

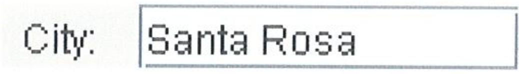

City: Santa Rosa

fig1.2

UI- Text Area

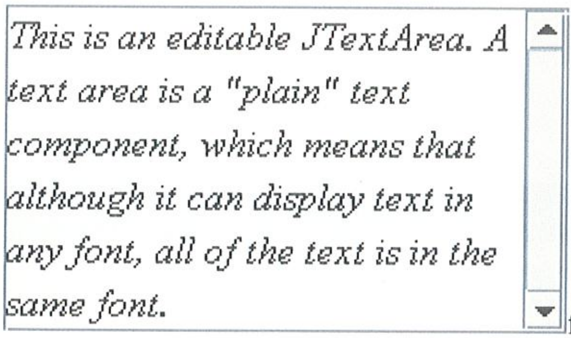

This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

fig1.3

UI- Password Field

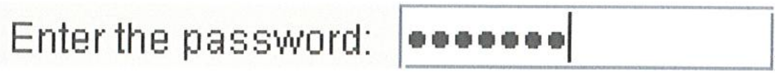
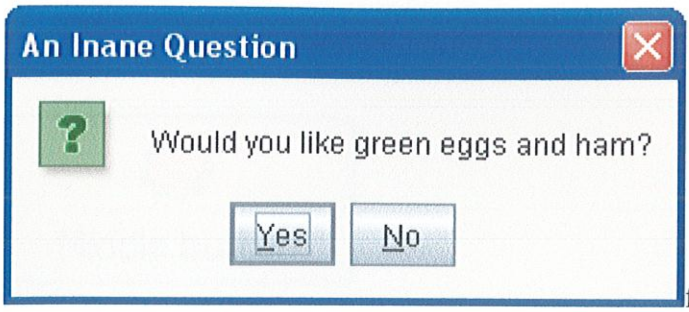

Enter the password: ●●●●●●●

fig1.4

UI- Dialogue Box



An Inane Question

Would you like green eggs and ham?

fig1.5

UI- Frame

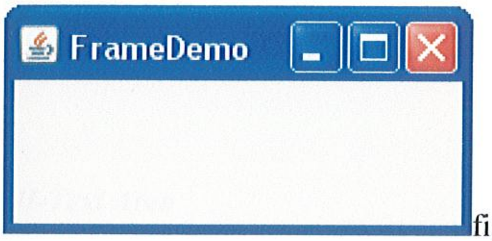
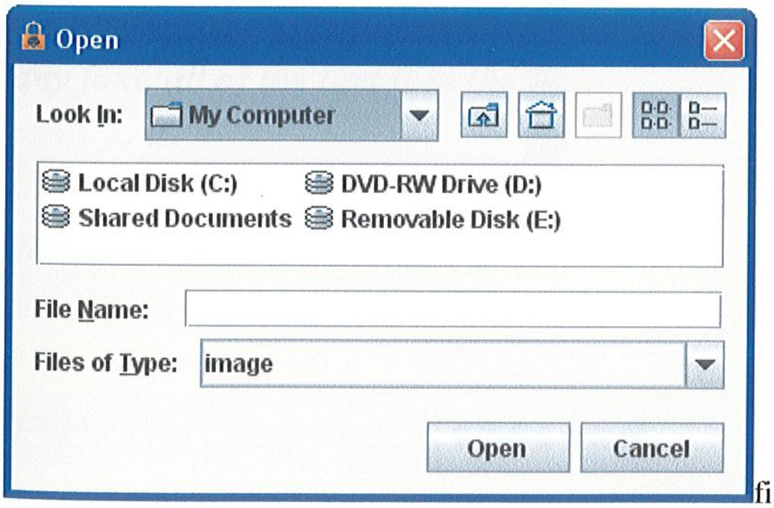


fig1.6

UI-File Chooser



Open

Look In: My Computer

Local Disk (C:)
DVD-RW Drive (D:)
Shared Documents
Removable Disk (E:)

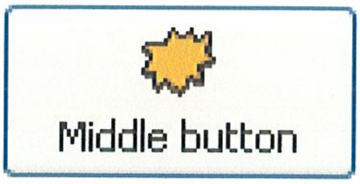
File Name:

Files of Type: image

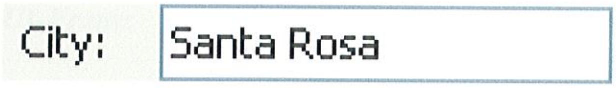
fig1.7

1.6 WINDOWS (LOOK AND FEEL)

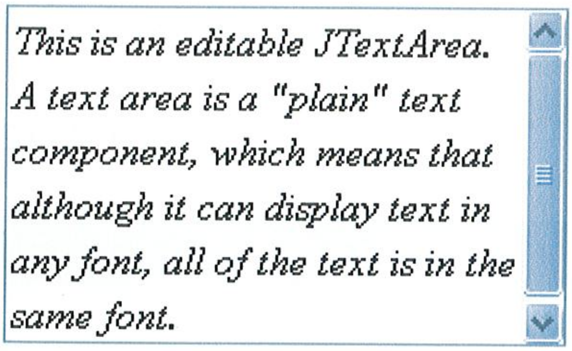
UI-Button



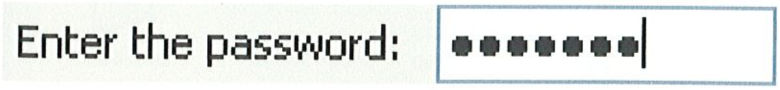
UI- Text Field


City: Santa Rosa

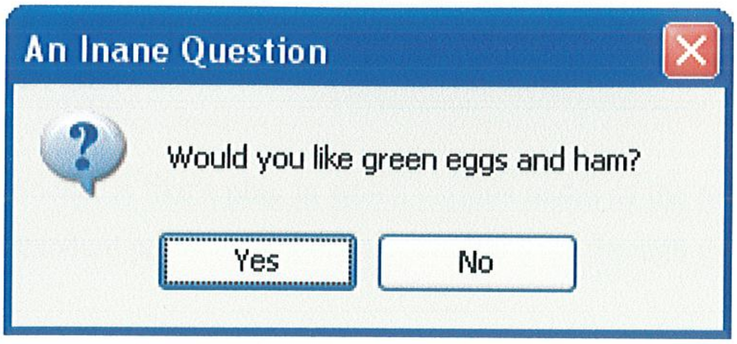
UI-Text Area


This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

UI-Password Field

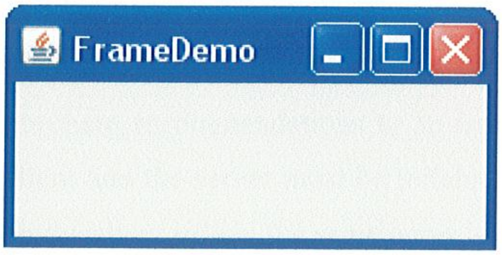

Enter the password: ●●●●●●●●

UI-Dialogue Box

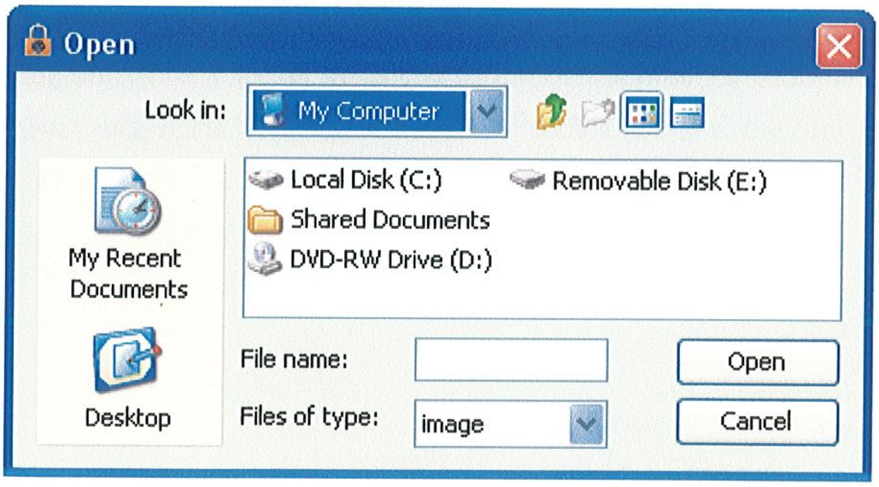


Would you like green eggs and ham?

UI-Frame



UI-File Chooser



Look in: My Computer

File name:

Files of type: image

CHAPTER 2
NETWORKING BASICS

2.1 SOCKETS

Socket is like a plug in which various nodes of the network are plugged in. Socket has a standard protocol, a node can communicate through the socket using that protocol.

URLS and URLConnections provide a relatively high-level mechanism for accessing resources on the Internet. Sometimes your programs require lower-level network communication, for example, when you want to write a client-server application.

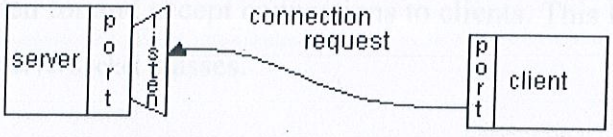
In client-server applications, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.

TCP provides a reliable, point-to-point communication channel that client-server application on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

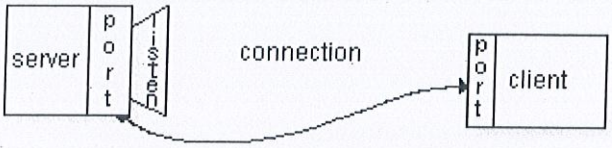
2.1.1Definition of a Socket

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

Definition: A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

Learning Resource Center
Acc. No. SP03020
JUIT, Waknaghat, Solan (H.P.)

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between your Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

Additionally, java.net includes the ServerSocket class, which implements a socket that servers can use to listen for and accept connections to clients. This lesson shows you how to use the Socket and ServerSocket classes.

If you are trying to connect to the Web, the URL class and related classes (URLConnection, URLEncoder) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation. See Working with URLs for information about connecting to the Web via URLs.

2.2 DIFFERENT TYPES OF SOCKETS

Sockets are classified according to communication properties. Processes usually communicate between sockets of the same type. However, if the underlying communication protocols support the communication, sockets of different types can communicate.

2.2.1 Stream Sockets

Provides sequenced, two-way byte streams with a transmission mechanism for stream data. This socket type transmits data on a reliable basis, in order, and with out-of-band capabilities.

With *stream sockets*, a process establishes a *connection* to another process. While the connection is in place, data flows between the processes in continuous *streams*. Stream sockets are said to provide a *connection-oriented service*. The protocol used for transmission is the popular *TCP (Transmission Control Protocol)*.

In the UNIX domain, the **SOCK_STREAM** socket type works like a pipe. In the Internet domain, the **SOCK_STREAM** socket type is implemented on the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol.

A *stream* socket provides for the bidirectional, reliable, sequenced, and unduplicated flow of data without record boundaries. Aside from the bidirectionality of data flow, a pair of connected stream sockets provides an interface nearly identical to pipes.

2.2.1 Datagram Sockets

They provide datagrams, which are connectionless messages of a fixed maximum length. This type of socket is generally used for short messages, such as a name server or time server, since the order and reliability of message delivery is not guaranteed.

With *datagram sockets*, individual *packets* of information are transmitted. This is not the right protocol for everyday users, because, unlike , the protocol used—*UDP*, the *User Datagram Protocol*—is a *connectionless service*, and it does not guarantee that packets arrive

In the UNIX domain, the **SOCK_DGRAM** socket type is similar to a message queue. In the Internet domain, the **SOCK_DGRAM** socket type is implemented on the User Datagram Protocol/Internet Protocol (UDP/IP) protocol.

A *datagram* socket supports the bidirectional flow of data, which is not sequenced, reliable, or unduplicated. A process receiving messages on a datagram socket may find messages duplicated or in an order different than the order sent. Record boundaries in data, however, are preserved. Datagram sockets closely model the facilities found in many contemporary packet-switched networks.

2.2.3Some other types of sockets

Table 2.1

SOCK_RAW Provides access to internal network protocols and interfaces. Available only to individuals with root user authority, a raw socket allows an application direct access to lower-level communication protocols. Raw sockets are intended for advanced users who wish to take advantage of some protocol feature that is not directly accessible through a normal interface, or who wishes to build new protocols atop existing low-level protocols.

Raw sockets are normally datagram-oriented, though their exact characteristics are dependent on the interface provided by the protocol.

SOCK_SEQPACKET Provides sequenced, reliable, and unduplicated flow of information.

SOCK_CONN_DGRAM Provides connection-oriented datagram service. This type of socket supports the bidirectional flow of data, which is sequenced and unduplicated, but is not reliable. Since this is a connection-oriented service, the socket must be connected prior to data transfer. Currently, only the Asynchronous Transfer Mode (ATM) protocol in the Network Device Driver (NDD) domain supports this socket type.

2.3 SOCKET PROTOCOLS

A *protocol* is a standard set of rules for transferring data, such as UDP/IP and TCP/IP. An application program can specify a protocol only if more than one protocol is supported for this particular socket type in this domain.

Each socket can have a specific protocol associated with it. This protocol is used within the domain to provide the semantics required by the socket type. Not all socket types are supported by each domain; support depends on the existence and implementation of a suitable protocol within the domain.

The **/usr/include/sys/socket.h** file contains a list of socket protocol families. These protocols are defined to be the same as their corresponding address families in the **socket** header file. Before specifying a protocol family, the programmer should check the **socket** header file for currently supported protocol families. Each protocol family consists of a set of protocols. Major protocols in the suite of Internet Network Protocols include:

- TCP
- UDP

2.3.1 *UDP Protocol:*

- UDP is a transport protocol --communication between processes
- UDP uses IP to deliver datagrams to the right host.
- UDP uses *ports* to provide communication services to individual processes.

Features:

- Datagram Delivery
- Connectionless
- Unreliable
- Minimal

2.3.2 *TCP (Transmission Control Protocol):*

- TCP provides:
- Connection-oriented
- Reliable
- Full-duplex
- Byte-Stream

Connection oriented means that a virtual connection is established before any user data is transferred. If the connection cannot be established - the user program is notified. If the connection is ever interrupted - the user program(s) is notified.

2.4 SERVER

A computer or device on a network that manages network resources. For example, a *file server* is a computer and storage device dedicated to storing files. Any user on the network can store files on the server. A *print server* is a computer that manages one or more printers, and a *network server* is a computer that manages network traffic. A database *server* is a computer system that processes database queries.

Servers are often dedicated, meaning that they perform no other tasks besides their server tasks. On multiprocessing operating systems, however, a single computer can execute several programs at once. A server in this case could refer to the program that is managing resources rather than the entire computer.

In general, server software has the following characteristics:

- It is a special-purpose program dedicated to providing one service.
- It is invoked automatically when a system boots, and continues to execute through many sessions.
- It runs on a shared computer.
- It waits passively for contact from arbitrary remote clients (*LISTEN* primitive).
- It accepts contact from arbitrary clients, but offers a single service.

2.4.1 Establishing a Simple Server Using Stream Sockets

Establishing a simple server in Java requires five steps.

Step 1) To create a ***Server-Socket*** object. A call to the ***ServerSocket*** constructor is made **ServerSocket server = new ServerSocket(** *port, queueLength* **);**

This *registers* an available *port number* and specifies a maximum number of clients that can wait to connect to the server (i.e., the *queueLength*). The port number is used by clients to located the server application on the server computer. This often is called the *handshake point*. If the queue is full, the server refuses client connections. The preceding statement establishes the port where the server waits for connections from clients (a process known as *binding the server to the port*). Each client will ask to connect to the server on this *port*. Programs manage each client connection with a ***Socket*** object. After binding the Server to a port with a **ServerSocket.**

Step 2) The server listens indefinitely (or *blocks*) for an attempt by a client to connect. To listen for a client, the program calls **Server-Socket** method **accept**, as in **Socket connection = server.accept();** This statement returns a **Socket** object when a connection with a client is established.

Step 3) is to get the **OutputStream** and **InputStream** objects that enable the server to communicate with the client by sending and receiving bytes. The server sends information to the client via an **OutputStream** object. The server receives information from the client via an **InputStream** object. To obtain the streams, the server invokes method **getOutputStream** on the **Socket** to get a reference to the **OutputStream** associated with the **Socket** and invokes method **getInputStream** on the **Socket** to get a reference to the **InputStream** associated with the **Socket**.
The **OutputStream** and **InputStream** objects can be used to send or receive individual bytes or sets of bytes with the **OutputStream** method **write** and the **InputStream** method **read**, respectively. Often it is useful to send or receive values of primitive data types (such as **int** and **double**) or **Serializable** class data types (such as **String**) rather than sending bytes. In this case, we can use the techniques of Chapter 16 to *chain* other stream types (such as **ObjectOutputStream** and **ObjectInputStream**) to the **OutputStream** and **InputStream** associated with the **Socket**.

For example,

ObjectInputStream input =
new ObjectInputStream(connection.getInputStream());

ObjectOutputStream output =
new ObjectOutputStream(connection.getOutputStream());

The beauty of establishing these relationships is that whatever the server writes to the **ObjectOutputStream** is sent via the **OutputStream** and is available at the client's **InputStream** and whatever the client writes to its **OutputStream** (with a corresponding **ObjectOutputStream**) is available via the server's **InputStream**.

Step 4) is the *processing* phase, in which the server and the client communicate via the **InputStream** and **OutputStream** objects. In Step 5, when the transmission is complete, the server closes the connection by invoking the **close** method on the **Socket** and on the corresponding streams.

2.5 CLIENT

The client part of a *client-server architecture*. Typically, a client is an application that runs on a personal computer or workstation and relies on a server to perform some operations. For example, an *e-mail client* is an application that enables you to send and receive e-mail.

In general, client software has the following characteristics:

- It is an application program that becomes a client temporarily when remote acces is needed, but performs other computation locally.
- It is invoked by a user and executes for one session.
- It runs locally on the user's computer.
- It actively initiates contact with a server (*CONNECT* primitive).
- It can access multiple services as needed.

2.5.1 Establishing a Simple Client Using Stream Sockets

Establishing a simple client in Java requires four steps.

Step 1) we create a **Socket** to connect to the server. The **Socket** constructor establishes the connection to the server. For example, the statement

Socket connection = new Socket(*serverAddress, port* **);**

uses the **Socket** constructor with two arguments—the server's Internet address (*serverAddress*) and the *port* number. If the connection attempt is successful, this statement returns a **Socket**. A connection attempt that fails throws an instance of a subclass of **IOException**, so many programs simply catch **IOException**.

An ***UnknownHostException*** occurs when a server address indicated by a client cannot be resolved. A ***ConnectException*** is thrown when an error occurs while attempting to connect to a server.

Step 2) The client uses **Socket** methods **getInputStream** and **getOutputStream** to obtain references to the **Socket**'s **InputStream** and **OutputStream**. As we mentioned in the preceding section, often it is useful to send or receive values of primitive data types (such as **int** and **double**) or class data types (such as **String** and **Employee**) rather than sending bytes. If the server is sending information in the form of actual data types, the client should receive the information in the same format.

Thus, if the server sends values with an **ObjectOutputStream**, the client should read those values with an **ObjectInputStream**.

Step 3) is the processing phase in which the client and the server communicate via the **InputStream** and **OutputStream** objects.

Step 4) the client closes the connection
when the transmission is complete by invoking the **close** method on the **Socket** and the corresponding streams. When processing information sent by a server, the client must determine when the server is finished sending information so the client can call **close** to

close the **Socket** connection. For example, the **InputStream** method **read** returns the value –1 when it detects end-of-stream (also called EOF—end-of-file).

If an **ObjectInputStream** is used to read information from the server, an **EOFException** occurs when the client attempts to read a value from a stream on which end-of-stream is detected.

2.6 INTERACTION BETWEEN CLIENT AND SERVER:

Like most application programs, a client and a server use a transport protocol to communicate. Figure 1 illustrates a client and a server using the TCP/IP protocol stack.

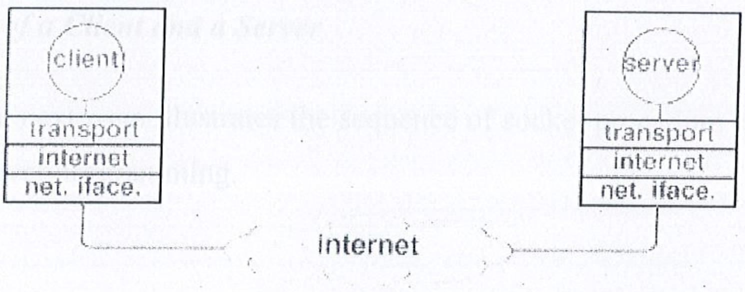


Fig 2.1

The client and server each interact with a protocol in the transport layer of the stack. A sufficiently powerful computer can run multiple servers and clients at the same time. Such a computer must have the necessary hardware resources (e.g. a fast CPU and sufficient memory) and have an operating system capable of running multiple applications concurrently (e.g. UNIX or Windows). Figure 2 illustrates such a setup.

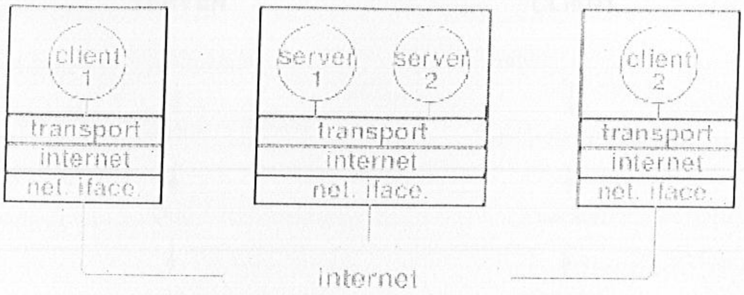


Fig 2.2 two servers

The computer in the middle might be running an FTP server and a WWW server. Modern computers are often capable of running many servers at the same time.

2.6.1 Examples of a Client and a Server

Figure 2.3 on the next page illustrates the sequence of socket procedure calls required for correct client-server programming.

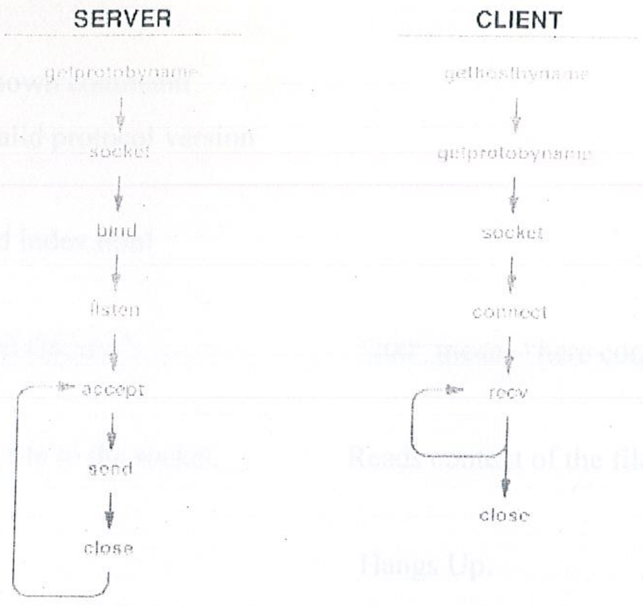


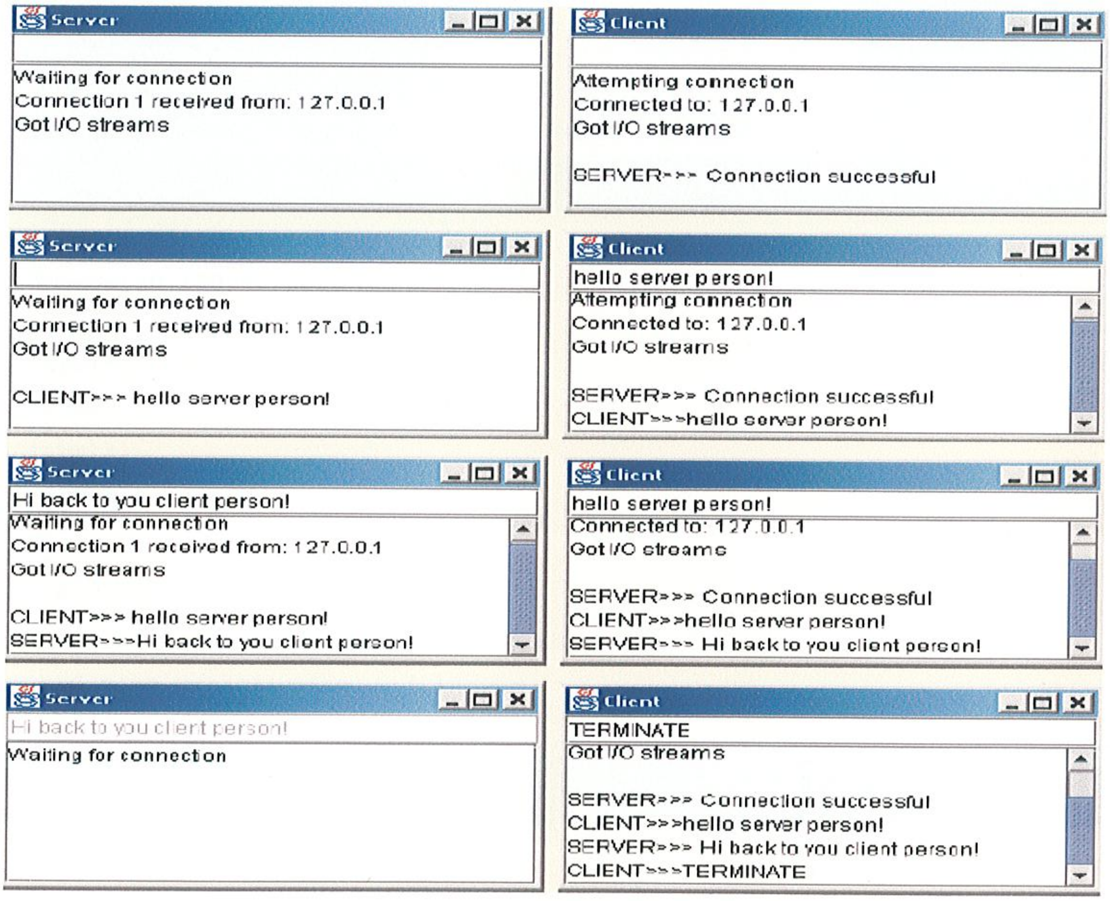
Fig 2.3: Socket procedure calls

Client and server applications can use either connection-oriented or connectionless transport protocols to communicate.

Server	*Client*
Listens to port 80.	Connects to port 80.
Accepts the connection.	Writes "GET/index.html HTTP/1.0\n\n".
Reads up until the second End-of-Line (\n)	

Sees that GET is a known command And HTTP/1.0 is a valid protocol version	
Reads local file called index.html	
Writes "HTTP/1.0 200 OK\n\n"	"200" means "here comes the file".
Copies content of the file to the socket.	Reads content of the file and displays it..
Hangs UP.	Hangs Up.

Fig 2.4



```
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams
```

```
Attempting connection
Connected to: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
```

```
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams

CLIENT>>> hello server person!
```

```
hello server person!
Attempting connection
Connected to: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
CLIENT>>>hello server person!
```

```
Hi back to you client person!
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams

CLIENT>>> hello server person!
SERVER>>>Hi back to you client person!
```

```
hello server person!
Connected to: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
CLIENT>>>hello server person!
SERVER>>> Hi back to you client person!
```

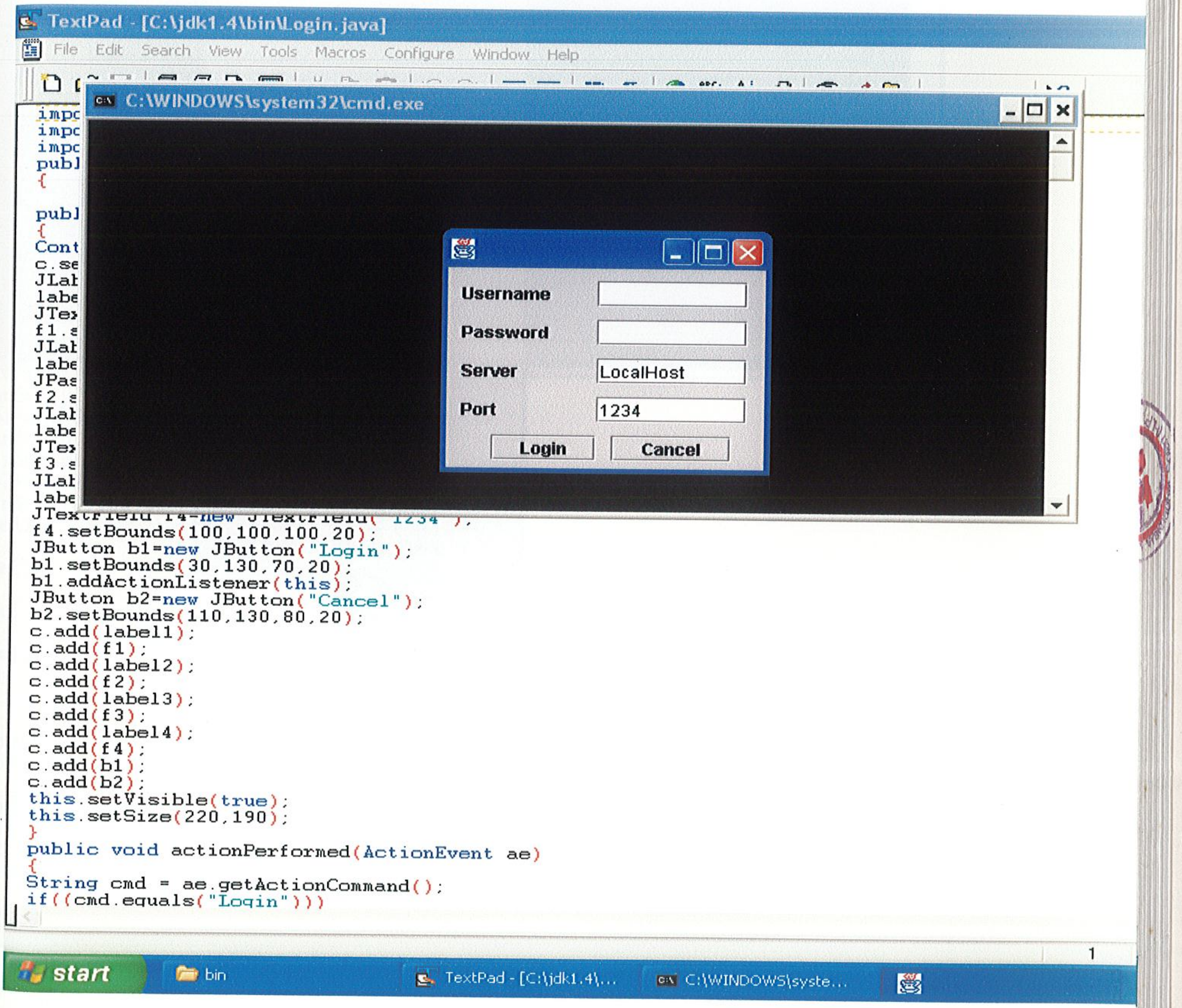
```
Hi back to you client person!
Waiting for connection
```

```
TERMINATE
Got I/O streams

SERVER>>> Connection successful
CLIENT>>>hello server person!
SERVER>>> Hi back to you client person!
CLIENT>>>TERMINATE
```

2.7 FEW SNAPSHOTS OF THE PROJECT

Login window

Fig 2.5



TextPad - [C:\jdk1.4\bin\Login.java]
File Edit Search View Tools Macros Configure Window Help
C:\WINDOWS\system32\cmd.exe
Login
Cancel
start
bin
TextPad - [C:\jdk1.4\...
C:\WINDOWS\syste...

Username

Password

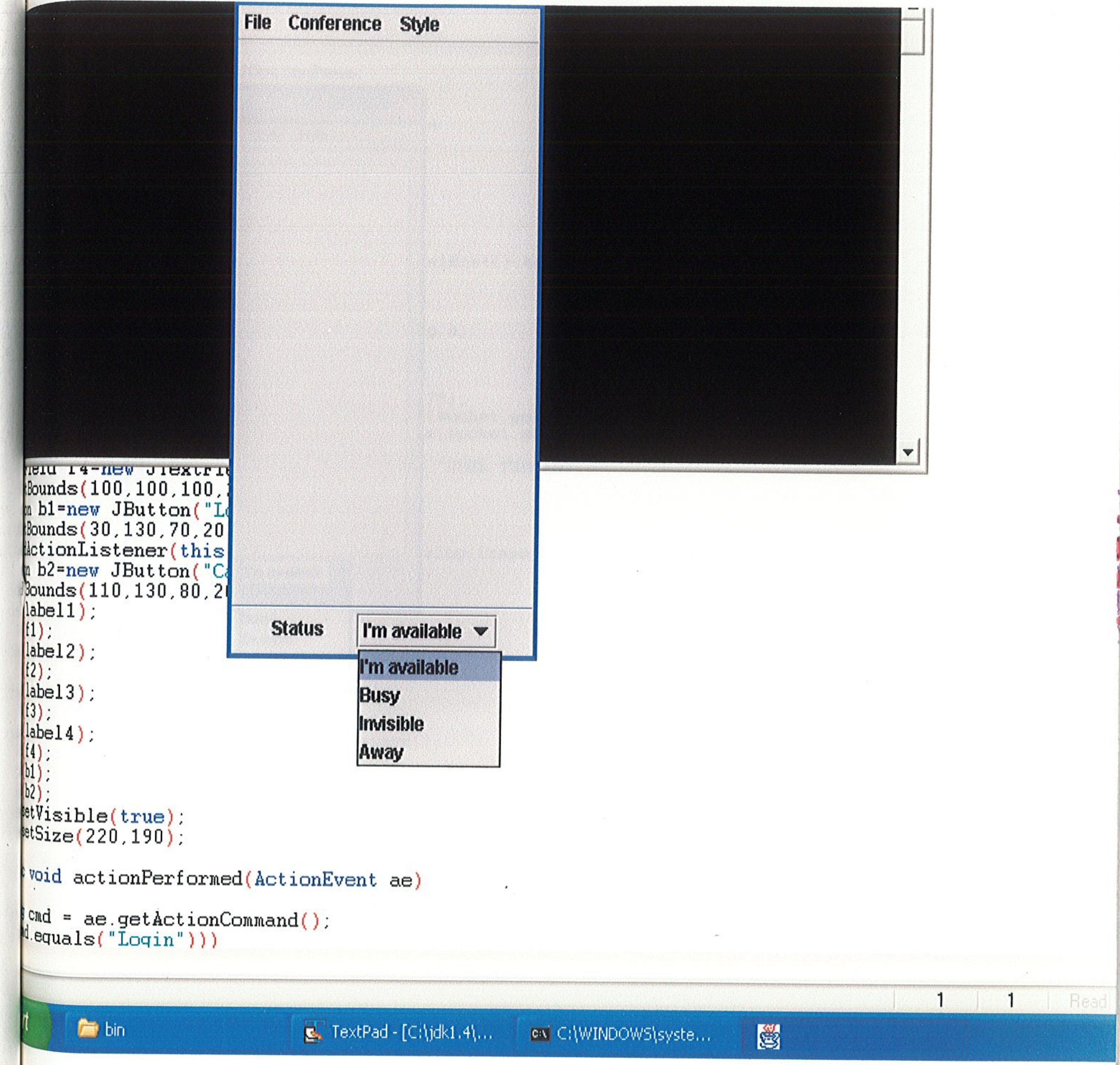
Server LocalHost

Port 1234

```
impc
impc
impc
publ
{

publ
{
Cont
c.se
JLak
labe
JTex
f1.s
JLak
labe
JPas
f2.s
JLak
labe
JTex
f3.s
JLak
labe
JTextField f4=new JTextField("1234");
f4.setBounds(100,100,100,20);
JButton b1=new JButton("Login");
b1.setBounds(30,130,70,20);
b1.addActionListener(this);
JButton b2=new JButton("Cancel");
b2.setBounds(110,130,80,20);
c.add(label1);
c.add(f1);
c.add(label2);
c.add(f2);
c.add(label3);
c.add(f3);
c.add(label4);
c.add(f4);
c.add(b1);
c.add(b2);
this.setVisible(true);
this.setSize(220,190);
}
public void actionPerformed(ActionEvent ae)
{
String cmd = ae.getActionCommand();
if((cmd.equals("Login")))
```

User window

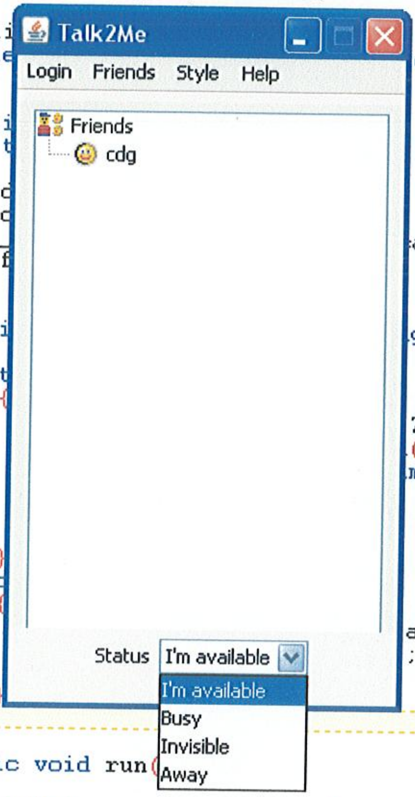
Fig. 2.6



```
ield f4=new JTextFie
Bounds(100,100,100,
n b1=new JButton("Lo
Bounds(30,130,70,20
ActionListener(this
n b2=new JButton("Ca
Bounds(110,130,80,2
label1);
f1);
label2);
f2);
label3);
f3);
label4);
f4);
b1);
b2);
etVisible(true);
etSize(220,190);

void actionPerformed(ActionEvent ae)

cmd = ae.getActionCommand();
d.equals("Login")))
```

Client

Fig 2.7

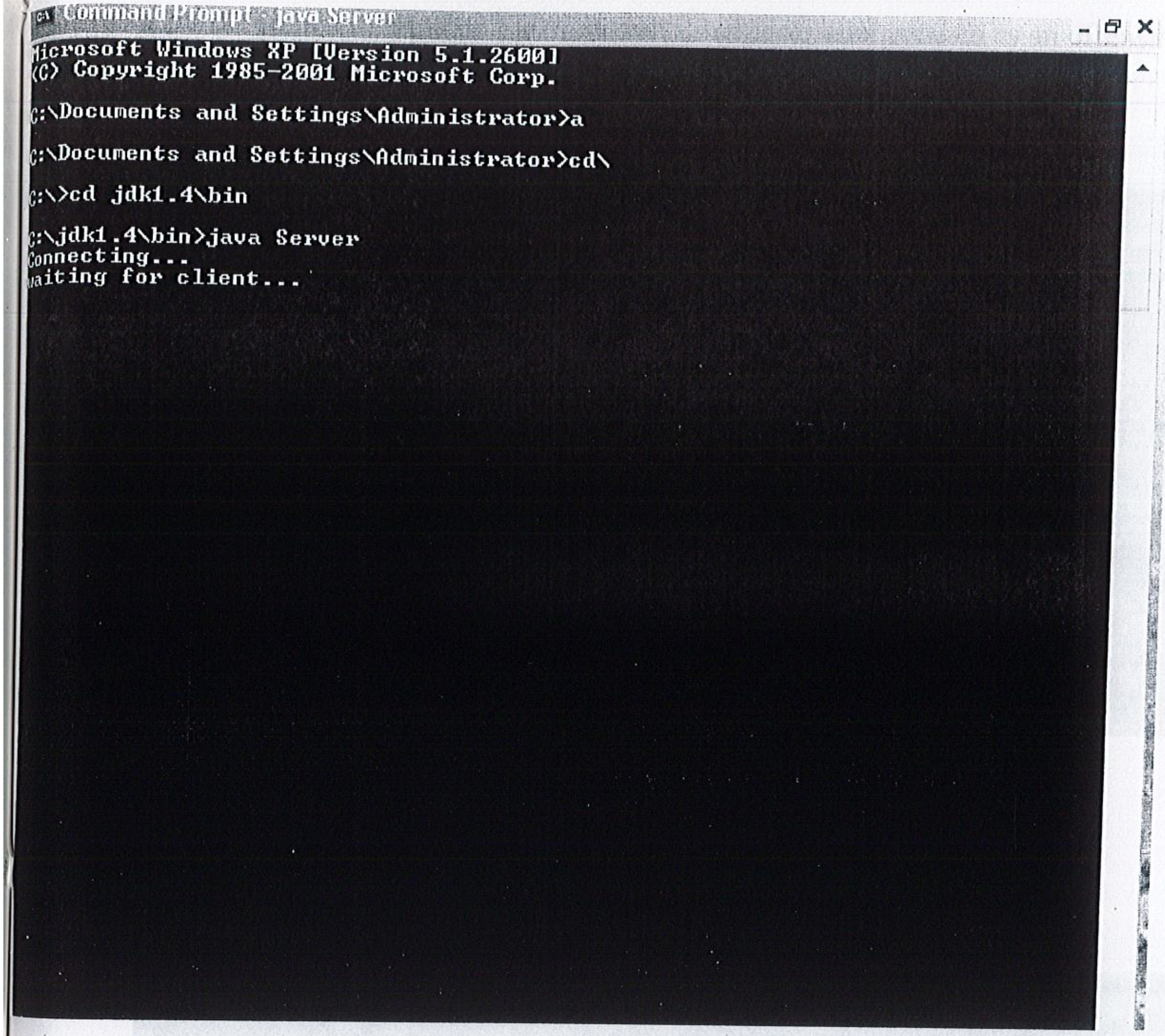


Status I'm available

```
import java.net.*;
import javax.swing.JOptionPane;

class Cli
    imple                                       e
{

    publi
        t
    {
        c
        c
                                                alHost().toString();
        f
    }

    publi                                       g s)
    {
        t
        {
                                                79);
                                                (socket.getInputStream());
                                                m(socket.getOutputStream());
                                                 "USER THREAD");

        }
        c
        {
                                                alog(frame, "Cannot find server !! Please
                                                ;

        }
    }

    public void run(
    {
        while(connected && !done)
            try
            {
                processServerMessage();
            }
```

Server

Fig 2.7

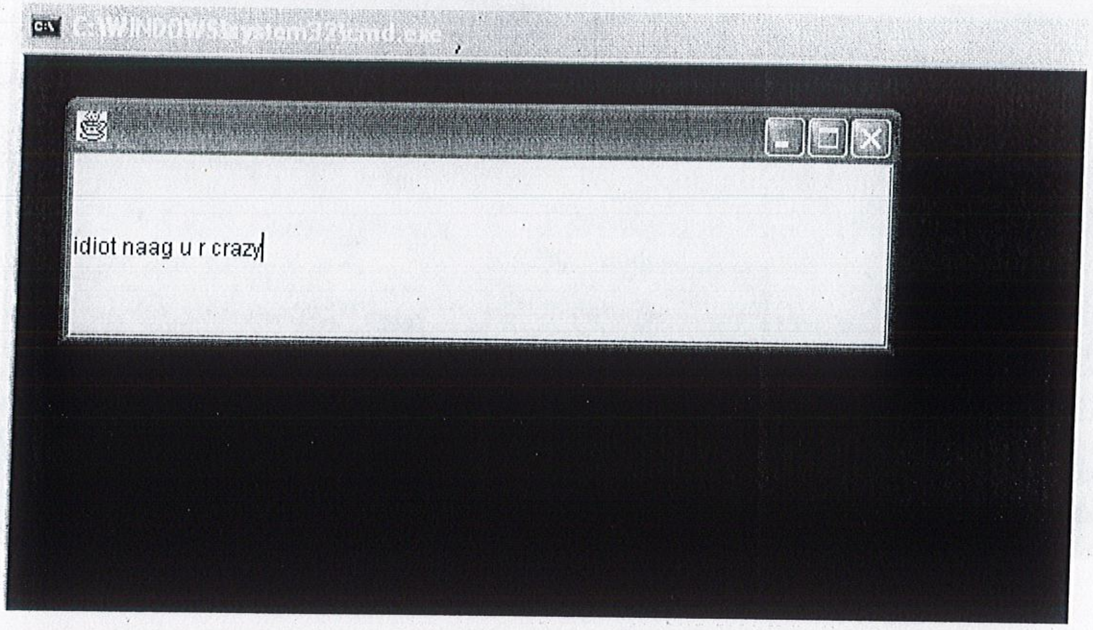


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>a

C:\Documents and Settings\Administrator>cd\

C:\>cd jdk1.4\bin

C:\jdk1.4\bin>java Server
Connecting...
Waiting for client...
```

Handling offensive words

Fig 2.8

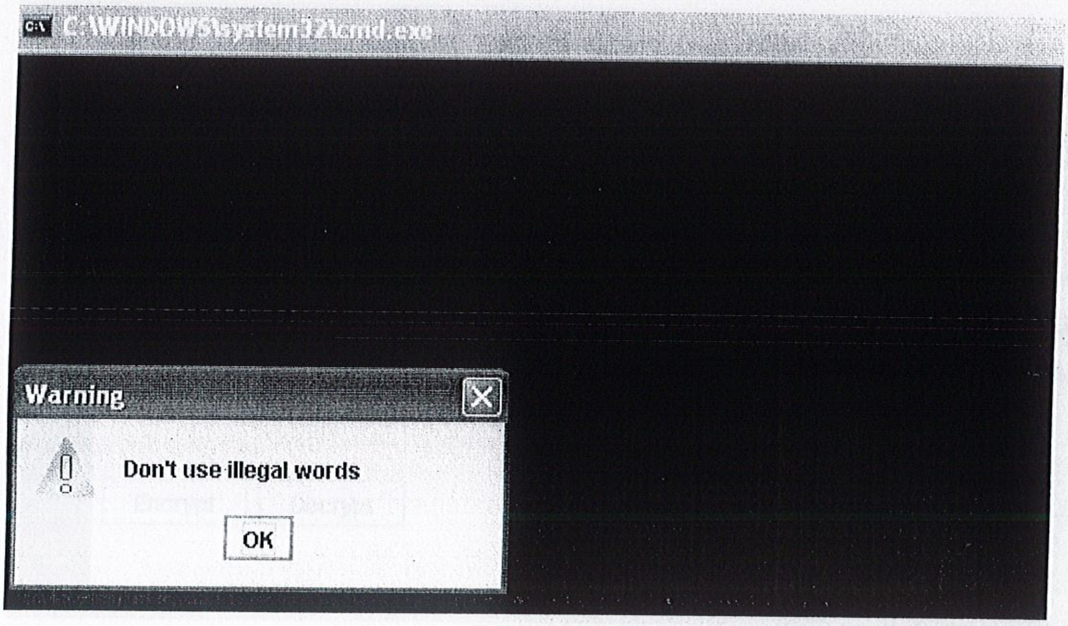
1) An illegal sentence: a client writes an illegal sentence that contains some harsh words which are not acceptable for example in this case idiot is taken as an offensive word



idiot naag u r crazy

2) Shows a warning: the system warns the user against using such words.

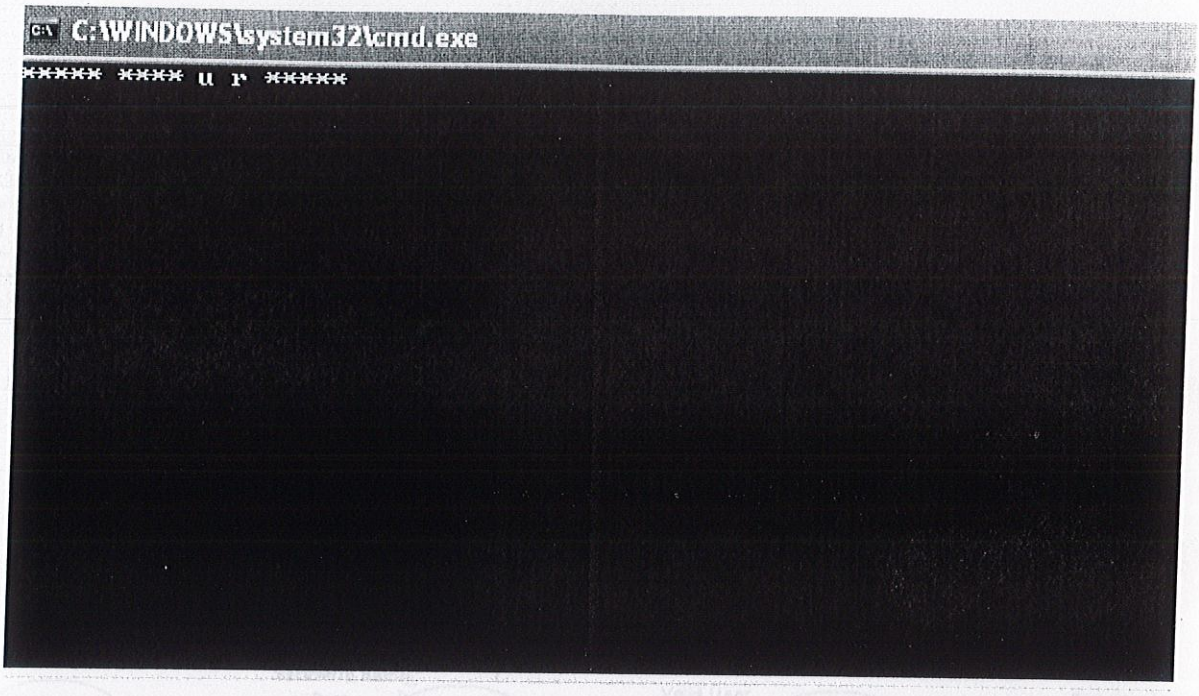
Fig 2.9



Don't use illegal words

3) Omits the illegal words: offensive words are omitted and the rectified message given.

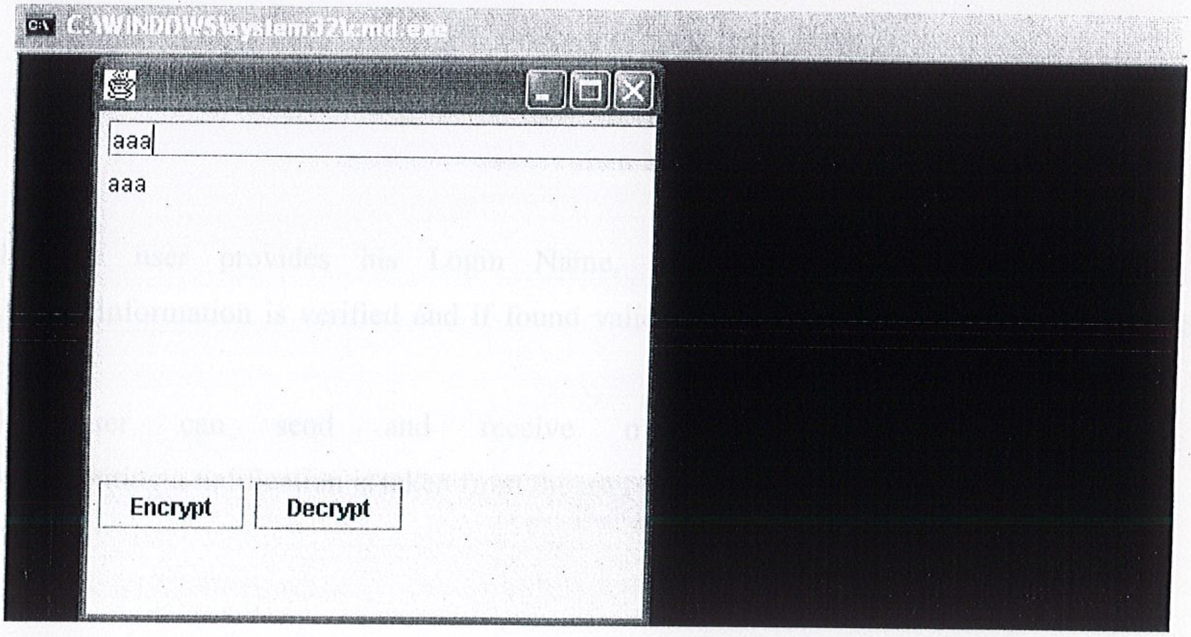
Fig 2.10



```
C:\WINDOWS\system32\cmd.exe
***** **** u r *****
```

Encryption Decryption

Fig 2.12



```
C:\WINDOWS\system32\cmd.exe
aaa
aaa
```

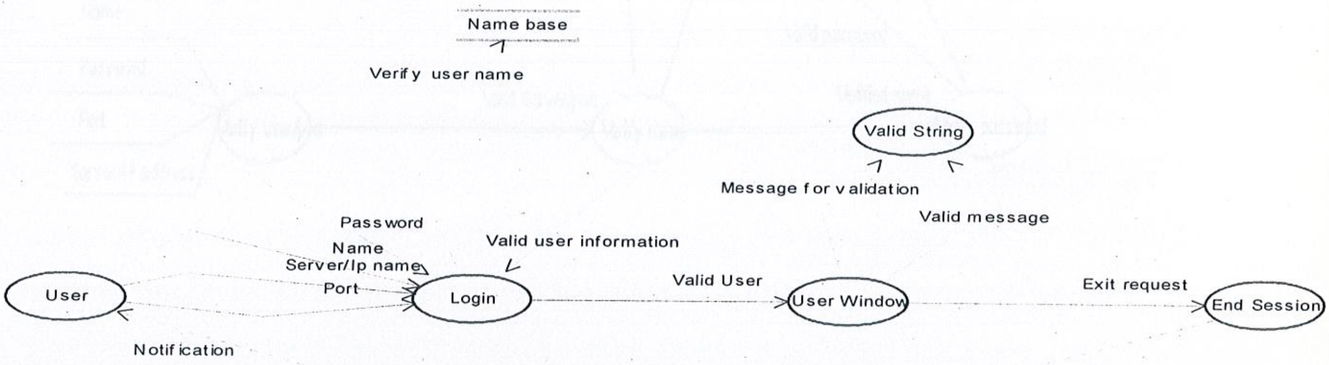
CHAPTER 3

MODELLING DESIGN AND FUNCTIONS USED

3.1 DATA FLOW DIAGRAM

3.1.1 Level 0 Data flow diagram

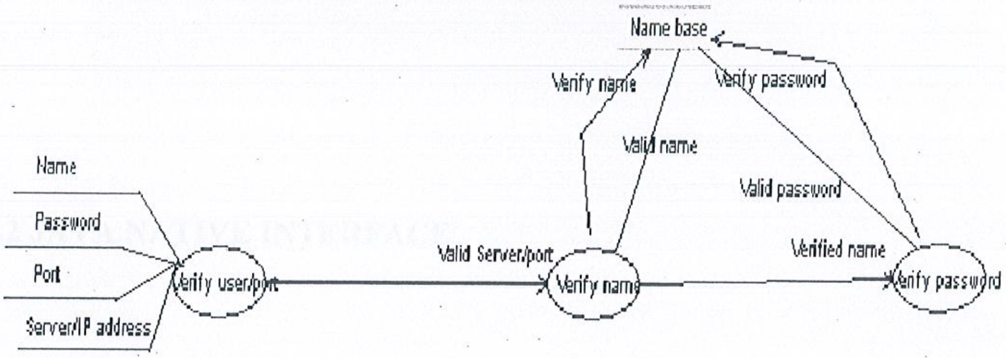
Fig 3.1 level 0 dataflow diagram



1) The user provides his Login Name, password, port number as inputs.
2) The information is verified and if found valid and authentic the user logs in to the client window.
3) User can send and receive messages while he is online.
4) On exiting a notification is taken from the user whether he really wants to exit.

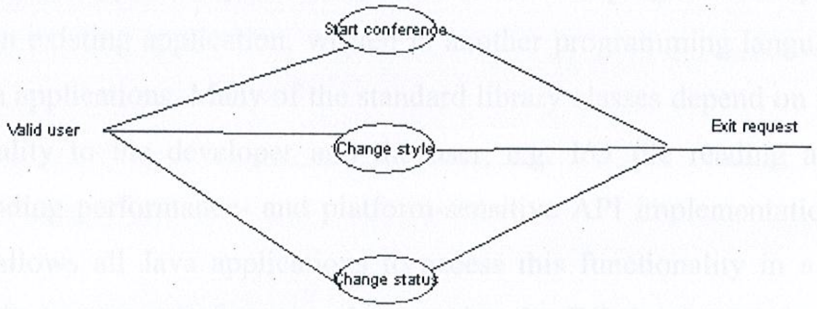
3.1.2 Login function

Fig 3.2 level 1 data flow diagram (login function)



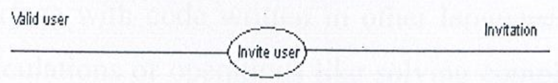
3.1.3 User window

Fig.3.3 User window dataflow diagram



3.1.4 Conference Data flow diagram

Fig. 3.4



3.2 JAVA NATIVE INTERFACE

The **Java Native Interface (JNI)** is a programming framework that allows Java code running in the Java virtual machine (VM) to call and be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages, such as C, C++ and assembly.

The JNI is used to write native methods to handle situations when an application cannot be written entirely in the Java programming language such as when the standard Java class library does not support the platform-specific features or program library. It is also used to modify an existing application, written in another programming language, to be accessible to Java applications. Many of the standard library classes depend on the JNI to provide functionality to the developer and the user, e.g. I/O file reading and sound capabilities. Including performance- and platform-sensitive API implementations in the standard library allows all Java applications to access this functionality in a safe and platform-independent manner. Before resorting to using the JNI developers should make sure the functionality is not already provided in the standard libraries.

The JNI framework lets a native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code.

JNI is sometimes referred to as the "escape valve" for Java developers because it allows them to add functionality to their Java Application that the Java API can't provide. It can be used to interface with code written in other languages, like C++. It is also used for time-critical calculations or operations like solving complicated mathematical equations, since native code can potentially be faster than JVM code.

The JNI is not trivial and requires a considerable effort to learn, and some people recommend that only advanced programmers should use the JNI. However, the capability for Java to communicate with C++ and assembly removes any limitations on what function Java programs can perform. Programmers considering using the JNI should be aware that

1. As mentioned before, the JNI is not an easy API to learn;
2. Only applications and signed applets can invoke the JNI;
3. An application that relies on JNI loses the platform portability Java offers (a workaround is to write a separate implementation of the JNI code for each platform and have Java detect the Operating System and load the correct one at runtime);
4. There is no garbage collection for the JNI side (JNI code must do explicit deallocation);
5. Error checking is a MUST or it has the potential to crash the JNI side and the JVM.

3.2.1 How the JNI works:

In JNI, native functions are implemented in a separate .c or .cpp file. (C++ provides a slightly cleaner interface with JNI.) When the JVM invokes the function, it passes a JNIEnv pointer, a jobject pointer, and any Java arguments declared by the Java method. A JNI function may look like this:

```
JNIEXPORT void JNICALL Java_ClassName_MethodName

 (JNIEnv *env, jobject obj)

{

  //Implement Native Method Here

}
```

The *env* pointer is a structure that contains the interface to the JVM. It includes all of the functions necessary to interact with the JVM and to work with Java objects. Example JNI functions are converting native arrays to/from Java arrays, converting native strings to/from Java strings, instantiating objects, throwing exceptions, etc. Basically, anything that Java code can do, can be done using JNIEnv, albeit with considerably less ease.

For example, the following converts a Java string to a native string:

```
//C++ code

JNIEXPORT void JNICALL Java_ClassName_MethodName

 (JNIEnv *env, jobject obj, jstring javaString)

{

  //Get the native string from javaString

  const char *nativeString = env->GetStringUTFChars(javaString, 0);


  //Do something with the nativeString
  //DON'T FORGET THIS LINE!!!

  env->ReleaseStringUTFChars(javaString, nativeString);

}

//C code

JNIEXPORT void JNICALL Java_ClassName_MethodName

 (JNIEnv *env, jobject obj, jstring javaString)

{

 //Get the native string from javaString

 const char *nativeString = (*env)->GetStringUTFChars(env, javaString, 0);

 //Do something with the nativeString

 //DON'T FORGET THIS LINE!!!

 (*env)->ReleaseStringUTFChars(env, javaString, nativeString);

}
```

Note that C++ JNI code is cleaner than C JNI code because like Java, C++ uses object method invocation semantics. That means that in C, the env parameter is dereferenced using (*env)-> and env has to be explicitly passed to JNIEnv methods. In C++, the env parameter is dereferenced using env-> and the env parameter is implicity passed as part of the object method invocation semantics.

Native data types can be mapped to/from Java data types. For compound types such as objects, arrays and strings the native code must explicitly convert the data by calling methods in the JNIEnv.

3.3 SOME FUNCTIONS USED

1) InetAddress

InetAddress class encapsulates numerical IP addresses and domain name for an address.The Inet address class does not have visible constructors, so to create an Inet Address class object we use one of the available methods known as ***factory methods.*** Three commonly used Inet Address factory methods are:

- Static InetAddress getlocalhost()

  /* returns InetAddress object that represents the host */
- Static InetAddress getByName(string *hostName)*

  /* returns the InetAddress for the host name passed to it */

If the above methods are unable to resolve the host name they pass an *unknownHost Exception.*

- Static InetAddress[] get AllByName(string *hostname*)

  /* returns an array of InetAddresses that represent all of the addresses that a particular host name resolves to */

InetAddress can handle both Ipv4 and Ipv6 addresses.

2) Socket I/O stream

Socket.getInputStream( )

/* returns the input stream associated with the invoking socket */

Socket.getOutputStream ( )

/* returns the output stream associated with the invoking socket */

3) Server Socket

ServerSocket class is used to create servers that listen for local or remote client programmes to connect to them and on published ports. The constructors are:

- ServerSocket(int *port*)
  /* Creates server socket on the specified port with a queue length of 50 */
- ServerSocket(int *port*, int *maxqueue*)
  /* Creates a server socket on the specified port with a maximum queue length of *maxqueue* */
- ServerSocket (int *port*, int *maxqueue,* InetAddress *localAddress*)
  /* Creates a server socket on the specified port with a maximum queue length of *maxqueue.* On a multihomed host, *localaddress* specifies the IP address to which this socket binds.*/

4) Get Source

Object.getSource( )

/* Returns the source of the event mentioned , It is a method used by the Event Object */

5) Get Action Event:

Action event is generated when a button is pressed, a list item is double clicked ,or a menu item is selected .

String getActionCommand ( )

/* Returns the command name for invoking Action Event object */

For example when a button is pressed, an action event is generated that has a command name equal to the label on that button.

6) Substrings

We can a extract a substring using substring ( ) function. It has two forms:

- String substring (int *startIndex*)
  /* start index specifies the index at which the substring will begin, this form returns a copy of the substring that begins at start index and runs to the end of invoking string. */

- String substring (int *startIndex,* int *endIndex*)
  /* End index represents the stopping point. the string returned contains all characters from beginning index up to but not including the ending index */

7) Add Element:

Void addElement(Object *element*)
/* The object specified by the *element* is added to the vector */

8) Thread Creation

The easiest method to create a thread is to create a class that implements the Runnable interface. Runnable abstracts a unit of executable code. We can create a thread on any object that implements Runnable. To implement Runnable class needs only a single method known as:

Public Void run ( )

Thread defines several constructors the one most commonly used is

Thread (Runnable *threadOb* , String *threadName*) /* The *threadOb* is an instance of a class that implements the Runnable interface, name of the new thread is specified by *threadName* .

Thread.start( ) .

/* The new thread created starts running only when this procedure is called it executes a call to run ( ) */

9) Set Layout

Void setLayout(LayoutManager *layoutObj*)

/* LayoutManager interface is implemented by instance of a class called layout manager. The layout manager is set by this method. Ifno call to this method is made then default layout manager is used.*/

10) Set Bounds

setBounds( )

/* Determines shape and position of each component manually. It is defined by Component. */

11) Set Editable

We can control that whether contents of a text field may be modified by a user or not by calling this function.

boolean isEditable( )

/* Returns true if text may be changed and false if not*/

void setEditable(boolean *canEdit*)

/* If *canEdit* is true text may be changed otherwise if it is false text cannot be altered. */

12) Set Visible

void setVisible(boolean *visibleFlag*)

/* Frame window created will not be visible untilthis method is called and *visible flag* is true */

13) Length

int length( ) /* Returns the number of characters a string contains i.e its length.

14) Add Type Listener:

Public void add*Type*Listener(*Type*Listener *el*)

/* *Type* is the name of the event and *el* is the reference to event listener , this method notifies listeners about specific types of events*/

CHAPTER 4

FUTURE SCOPES AND TESTING

4.1 FUTURE SCOPES

1) Encryption in a Network.
2) Voice Chat between two clients over the Network.
3) Logging on to any remote PC on the Network for accessing files

4.1. ENCRYPTION IN NETWORK

We intend to build a chat server in which secured chatting between two clients could take place. That is personal information sent by a client to another should be in an encrypted form and could be decrypted only by the client who has the key. The key can be sent to the second client through a personal message. We basically use the same key to both Encrypt and Decrypt data, this Technique is called Encryption and Decryption using symmetric keys.

4.2 ENCRYPTION AND DECRYPTION USING SYMMETRIC KEYS

Encryption and decryption can be done symmetrically -- here the same key is used to encrypt and decrypt the data. Because both parties have the same key, the decryption essentially is performed by reversing some part of the encryption process. The Blowfish algorithm is an example of a symmetric key. It is supported by the Java Cryptography Extension (JCE). You can find the appropriate APIs in the javax.crypto.* packages. In addition to Blowfish, examples of cipher algorithms currently supported by the JCE are the Digital Encryption Standard (DES), Triple DES Encryption (DESede), and Password-based encryption algorithm (PBEWithMD5AndDES).

Symmetric key algorithms tend to be much faster than asymmetric key algorithms. In addition, as you saw in the first tip, the size of the text that can be encrypted depends on the size of the product of the two primes used to generate the public and private keys. With symmetric key algorithms you do not have a limitation on the total size of what can be encrypted. Although, depending on the symmetric cipher algorithms, the total input size has to be a multiple of block sizes and might require padding. A problem with symmetric keys is that keys must be shared among parties involved in encryption or decryption. So there is the danger of interception or unauthorized sharing.

We create a symmetric key it is much as creating a key pair. A factory method from the KeyGenerator class is used and passed in the algorithm as a String. When one calls the generateKey () method, one gets back an object that implements the Key interface instead of the KeyPair interface. The call looks something like this:

```
SecretKey key =
    KeyGenerator.getInstance ("DES").generateKey ();
```

Next we need to create a Cipher. This is the workhorse for JCE. We again use a factory method of the Cipher class so that we can take advantage of different providers without changing the application. Cipher is created like this:

```
Cipher cipher = Cipher.getInstance("DES");
```

A Cipher is used to encrypt and decrypt data that is passed in as byte arrays. The two essential methods you must use are

1) init ( ) /* to specify which operation will be called */
and
2) doFinal() /* to perform the operation*/

. ***For example***, the following two lines use the cipher and key instances created to encrypt a byte array called textBytes. The result is stored in a byte array called encryptedBytes.

```
cipher.init(Cipher.ENCRYPT_MODE, key);
```

```
byte[] encryptedBytes =
   cipher.doFinal( textBytes );
```

4.2.1 Sample programs:

The following is a sample programme that takes an input string and encrypts it………..

```
import javax.crypto.Cipher;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import java.security.Key;
import java.security.InvalidKeyException;

public class LocalEncrypter {


    private static String algorithm = "DESede";
    private static Key key = null;
    private static Cipher cipher = null;


    private static void setUp() throws Exception {
        key = KeyGenerator.getInstance(algorithm).generateKey();
        cipher = Cipher.getInstance(algorithm);
    }

   public static void main(String[] args)
     throws Exception {
      setUp();
      if (args.length !=1) {
         System.out.println(
           "USAGE: java LocalEncrypter " +
                              "[String]");
         System.exit(1);
        }
        byte[] encryptionBytes = null;
        String input = args[0];
        System.out.println("Entered: " + input);
        encryptionBytes = encrypt(input);
        System.out.println(
          "Recovered: " + decrypt(encryptionBytes));
     }

   private static byte[] encrypt(String input)
        throws InvalidKeyException,
            BadPaddingException,
            IllegalBlockSizeException {
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] inputBytes = input.getBytes();
        return cipher.doFinal(inputBytes);
   }

   private static String decrypt(byte[] encryptionBytes)
        throws InvalidKeyException,
            BadPaddingException,
            IllegalBlockSizeException {
        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] recoveredBytes =
          cipher.doFinal(encryptionBytes);
        String recovered =
          new String(recoveredBytes);
        return recovered;
      }
```

One can enter any text as one likes as a command line parameter. For example, if we submit the following on the command line:

```
java LocalEncrypter "Whatever phrase we would like to
 input at this point"
```

We Will see something like this as output:

```
Entered: Whatever phrase we would like to input at this point
Recovered: Whatever phrase we would like to input at this point
```

In this example. both the encryption and the decryption were done with the same Key object. Encryption and decryption ordinarily occur on different VMs at different times, so you need a method for securely transporting the key.

4.3 PROJECT TESTING (black box testing)

Black Box testing refers to the technique of testing a system with no knowledge of the internals of the system. Black Box testers do not have access to the source code and are oblivious of the system architecture. A Black Box tester typically interacts with a system through a user interface by providing inputs and examining outputs without knowing where and how the inputs were operated upon. In Black Box testing, target software is exercised over a range of inputs and the outputs are observed for correctness.

Advantages

* Efficient Testing — Well suited and efficient for large code segments or units.

* Unbiased Testing — clearly separates user's perspective from developer's perspective through separation of QA and Development responsibilities.

* Non intrusive — code access not required.

* Easy to execute — can be scaled to large number of moderately skilled testers with no knowledge of implementation, programming language, operating systems or networks.

Disadvantages

* Localized Testing — Limited code path coverage since only a limited number of test inputs are actually tested.

* Inefficient Test authoring — without implementation information, exhaustive input coverage would take forever and would require tremendous resources.

* Blind Coverage — cannot control targeting code segments or paths which may be more error prone than others.

Black Box testing is best suited for rapid test scenario testing and quick Web Service prototyping. This testing technique for Web Services provides quick feedback on the functional readiness of operations through

The symbols followed in the design of the test cases are as under.

I: input present

S: input absent

X: don't care

P: output present

A: output absent

4.4 TEST CASES:

4.4.1 LOGIN FUNCTION

Inputs (causes):

C1: Server IP Supplied

C2: User Name Supplied

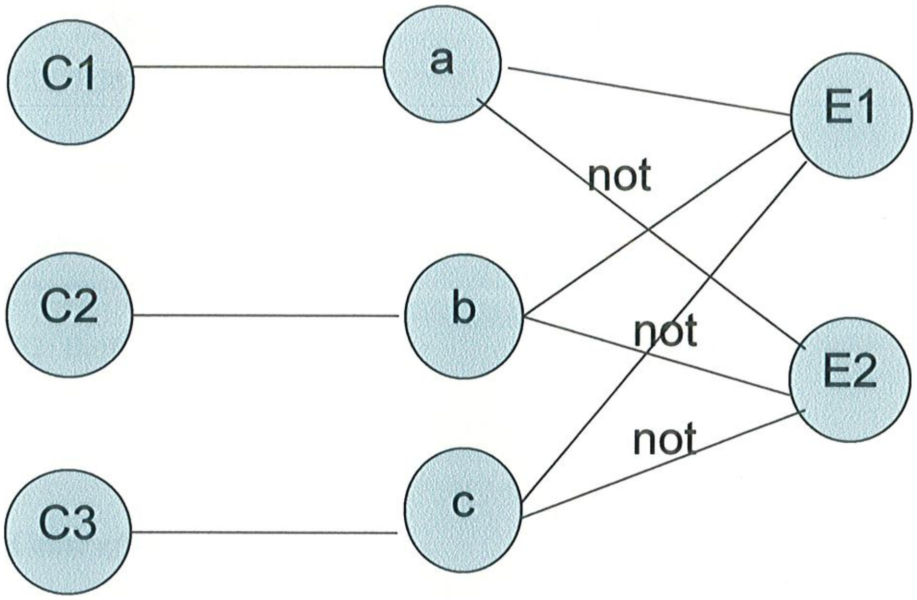
C3: Password Entered

Outputs (effects):

E1: Client Connected To Server, User Window Launched

E2: Client Not Connected To Server, Error Message

Cause effect graph Fig 4.1



	Test 1	Test 2	Test 3	Test 4	Test 5
C1	I	I	S	S	I
C2	I	S	I	S	I
C3	I	I	I	I	S
C4	P	A	A	A	A
C5	A	P	P	P	P

Table 4.1
Test Cases

4.4.2 FILE TRANSFER

Inputs (causes):

C1: File size = LOW

C2: File size = SAFE

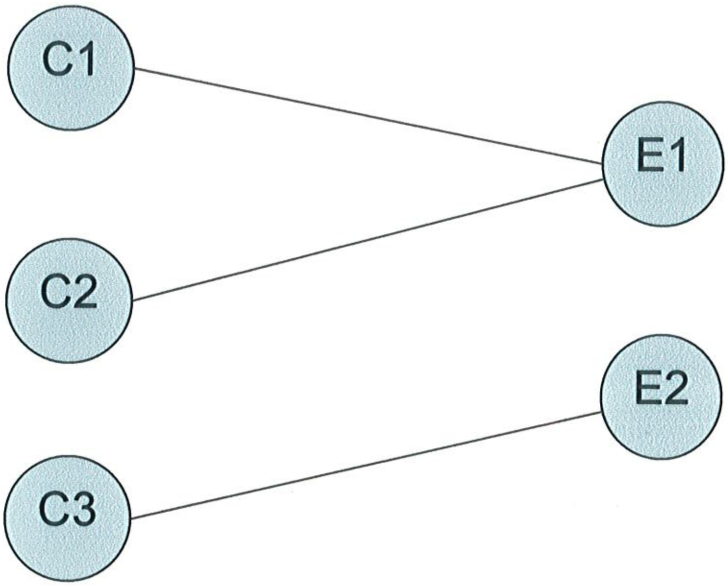
C3: File size = HIGH

Outputs (effects):

E1: File transferred

E2: Buffer Overflow

Cause effect graph Fig 4.2



Test cases

Table 4.2

	Test 1	Test 2	Test 3	Test 4
C1	I	I	S	S
C2	I	S	I	S
C3	S	S	S	I
C4	P	P	P	A
C5	A	A	A	P

5 SAMPLE CODES

Code1: For file selection

```
public class FileChooserDemo2 extends JPanel
                          implements ActionListener {
   static private String newline = "\n";
   private JTextArea log;
   private JFileChooser fc;
   Socket socket1;
   File file;
    Msg m1 = new Msg();
JFrame f =new JFrame();
   public FileChooserDemo2() {

       super(new BorderLayout());

       //Create the log first, because the action listener
       //needs to refer to it.
       log = new JTextArea(5,20);
       log.setMargin(new Insets(5,5,5,5));
       log.setEditable(false);
       JScrollPane logScrollPane = new JScrollPane(log);

       JButton sendButton = new JButton("Attach...");
          sendButton.addActionListener(this);

      add(sendButton, BorderLayout.PAGE_START);
      add(logScrollPane, BorderLayout.CENTER);
if (fc == null) {
        fc = new JFileChooser();

      }

     //Show it.
     int returnVal = fc.showDialog(FileChooserDemo2.this,
                              "Attach");

     //Process the results.
     if (returnVal == JFileChooser.APPROVE_OPTION) {
             file = fc.getSelectedFile();
      System.out.println("Attaching file: " + file.getName()
                 + "." + newline );
            log.append("Attaching file: " + file.getName()
                 + "." + newline);


        }
        log.setCaretPosition(log.getDocument().getLength());
             //Reset the file chooser for the next time it's shown.
        fc.setSelectedFile(null);

  Container c = f.getContentPane();
  c.add(this);
  f.setVisible(true);
  f.setSize(300,300);

     }

   public void actionPerformed(ActionEvent e)
  {
 //code for file transfer to be added
 }
 }
```

Code 2: Handles offensive words

```
class Match1
{
        protected JTextField j;
        static JFrame frame = new JFrame();
        public Match1()
        {

        }

public static String returnUnSlangged(String msg)
{
                String org=msg;
```

```
                    frame.setVisible(false);
                        String search[]={"words which are to be banned"}
                        String sub="*";
                        String result="";
                        int i;
                        int p=0;
                        //System.out.println(search.length);

                    do
                    {
                        do
                        {

                /* code for searching illegal words from the message sent by
one client to another.*/
                        }while(p<search.length);
            System.out.println(org);

        return org;
}
```

Code 3: Handles each client individually

```
class Service
  implements Runnable
{

  public Service(Socket socket1, String s, User user1)
  {
    done = false;
    try
    {
      socket = socket1;
      hostname = s;
      user = user1;
```

```
            dis = new DataInputStream(socket.getInputStream());
            thread = new Thread(this, "SERVICE");
            thread.start();
        }
        catch(Exception exception)
        {
            System.out.println("service constructor" + exception);
        }
    }

    public void run()
    {
        while(!done)
            try
            {
                byte abyte0[] = new byte[2048];
                dis.read(abyte0);
                Message message = (Message)ChatUtils.bytesToObject(abyte0);
                Server.processClientMessage(message);

            }
            catch(Exception exception)
            {
                done = true;
                Server.removeUser(user);
                Message message1 = new Message(2);
                user.isOnline = 2;
                message1._user = user;
                Server.writeToClients(message1);
                try
                {
                    socket.close();
                }
                catch(Exception exception1)
                {
                    System.out.println("ERROR CLOSING SOCKET " + exception1);
                }
            }
    }
}
```

CONCLUSION

"One should aim for the stars, so that you can fall on the clouds at least".

The basic application of chatting software is to enable reliable and correct transferring of messages between two clients. Our Chat Server implements that basic application along with some interesting features like:

Omission of some offensive words from the message,

File transfer between clients which includes all file types.

In future we can also add encryption and decryption of a message to insure that secure message transferring can take place between two clients over the network.

Black Box Testing has been efficiently used to test the software developed as the software is concerned to educate the user about the good and faulty system behavior. Actually, Black Box testing deals with just the input and output which is what the user is interested in.

BIBLIOGRAPHY

Websites referred

- Google.com
- Java.sun.com
- Wikipedia.org
- Forums

Books referred

- The Complete Reference by Herbert Schildt.
- Java I/O by O'Reilly