

## **1. DEVELOP AN FTP CLIENT. PROVIDE A GUI INTERFACE FOR THE ACCESS OF ALL THE SERVICES.**

### **FTP:**

File Transfer Protocol (FTP) is a standard network protocol used to copy a file from one host to another over a TCP/IP-based network, such as the Internet. FTP is built on a client-server architecture and utilizes separate control and data connections between the client and server.[1] FTP users may authenticate themselves using a clear-text sign-in protocol but can connect anonymously if the server is configured to allow it.

The first FTP client applications were interactive command-line tools, implementing standard commands and syntax. Graphical user interface clients have since been developed for many of the popular desktop operating systems in use today.

A client makes a TCP connection to the server's port 21. This connection, called the control connection, remains open for the duration of the session, with a second connection, called the data connection, opened by the server from its port 20 to a client port (specified in the negotiation dialog) as required to transfer file data. The control connection is used for session administration (i.e., commands, identification, passwords) exchanged between the client and server using a telnet-like protocol. For example "RETR filename" would transfer the specified file from the server to the client. Due to this two-port structure, FTP is considered an out-of-band, as opposed to an in-band protocol such as HTTP.

The server responds on the control connection with three digit status codes in ASCII with an optional text message, for example "200" (or "200 OK.") means that the last command was successful. The numbers represent the code number and the optional text represent explanations (i.e., <OK>) or needed parameters (i.e., <Need account for storing file>). A file transfer in progress over the data connection can be aborted using an interrupt message sent over the control connection.

FTP can be run in active or passive mode, which determine how the data connection is established. In active mode, the client sends the server the IP address and port number on which the client will listen, and the server initiates the TCP connection. In situations where the client is behind a firewall and unable to accept incoming TCP connections, passive mode may be used. In this mode the client

sends a PASV command to the server and receives an IP address and port number in return. The client uses these to open the data connection to the server. Both modes were updated in September 1998 to add support for IPv6. Other changes were made to passive mode at that time, making it extended passive mode.

While transferring data over the network, four data representations can be used:

- \* ASCII mode: used for text. Data is converted, if needed, from the sending host's character representation to "8-bit ASCII" before transmission, and (again, if necessary) to the receiving host's character representation. As a consequence, this mode is inappropriate for files that contain data other than plain text.

- \* Image mode (commonly called Binary mode): the sending machine sends each file byte for byte, and the recipient stores the byte-stream as it receives it. (Image mode support has been recommended for all implementations of FTP).

- \* EBCDIC mode: use for plain text between hosts using the EBCDIC character set. This mode is otherwise like ASCII mode.

- \* Local mode: Allows two computers with identical setups to send data in a proprietary format without the need to convert it to ASCII. For text files, different format control and record structure options are provided. These features were designed to facilitate files containing Telnet or ASA formatting.

Data transfer can be done in any of three modes[1]:

- \* Stream mode: Data is sent as a continuous stream, relieving FTP from doing any processing. Rather, all processing is left up to TCP. No End-of-file indicator is needed, unless the data is divided into records.

- \* Block mode: FTP breaks the data into several blocks (block header, byte count, and data field) and then passes it on to TCP.

- \* Compressed mode: Data is compressed using a single algorithm (usually Run-length encoding).

### **List of FTP commands:**

Below is a **list of FTP commands** that may be sent to an FTP server, including all commands that are standardized in RFC 959 by the IETF. All commands below are RFC 959 based unless stated otherwise. Note that most command-line FTP clients present their own set of commands to users. For example, GET is the common user command to download a file instead of the raw command RETR.

Command ☒	Description
ABOR	Abort an active file transfer.
ACCT	Account information.
ADAT	Authentication/Security Data
ALLO	Allocate sufficient disk space to receive a file.
APPE	Append.
AUTH	Authentication/Security Mechanism
CCC	Clear Command Channel
CDUP	Change to Parent Directory.
CONF	Confidentiality Protection Command
CWD	Change working directory.
DELE	Delete file.
ENC	Privacy Protected Channel
EPRT	Specifies an extended address and port to which the server should connect.
EPSV	Enter extended passive mode.
FEAT	Get the feature list implemented by the server.
HELP	Returns usage documentation on a command if specified, else a general help document is returned.
LANG	Language Negotiation
LIST	Returns information of a file or directory if specified, else information of the current working directory is returned.
LPRT	Specifies a long address and port to which the server should connect.
LPSV	Enter long passive mode.
MDTM	Return the last-modified time of a specified file.
MIC	Integrity Protected Command
MKD	Make directory.
MLSD	Lists the contents of a directory if a directory is named.
MLST	Provides data about exactly the object named on its command line, and no others.
MODE	Sets the transfer mode (Stream, Block, or Compressed).
NLST	Returns a list of file names in a specified directory.
NOOP	No operation (dummy packet; used mostly on keepalives).
OPTS	Select options for a feature.
PASS	Authentication password.
PASV	Enter passive mode.
PBSZ	Protection Buffer Size

PORT	Specifies an address and port to which the server should connect.
PROT	Data Channel Protection Level.
PWD	Print working directory. Returns the current directory of the host.
QUIT	Disconnect.
REIN	Re initializes the connection.
REST	Restart transfer from the specified point.
RETR	Transfer a copy of the file
RMD	Remove a directory.
RNFR	Rename from.
RNTO	Rename to.
SITE	Sends site specific commands to remote server.
SIZE	Return the size of a file.
SMNT	Mount file structure.
STAT	Returns the current status.
STOR	Accept the data and to store the data as a file at the server site
STOU	Store file uniquely.
STRU	Set file transfer structure.
SYST	Return system type.
TYPE	Sets the transfer mode (ASCII/Binary).
USER	Authentication username.

**ftpserver:**

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
public class ftpserver
{
public static void main(String args[])
{
try
{
ServerSocket ss=new ServerSocket(8020);
Socket connection=ss.accept();
System.out.println("SERVER SOCKET IS CREATED");
DataInputStream input=new DataInputStream(connection.getInputStream());
String option=input.readLine();

if(option.equalsIgnoreCase("Upload"))
{
System.out.println("Upload text");
String inputfile=input.readLine();
File clientfile=new File(inputfile);
FileOutputStream fout=new FileOutputStream(clientfile);
int ch;
while((ch=input.read())!=-1)
{
fout.write((char)ch);
}
fout.close();
}

if(option.equalsIgnoreCase("download"))
{
System.out.println("download text");
String filefromclient=input.readLine();
```

```

File clientfile=new File(filefromclient);
FileInputStream fis=new FileInputStream(clientfile);
PrintStream out=new PrintStream(connection.getOutputStream());
int n=fis.read();
while(n!=-1)
{
out.print((char)n);
n=fis.read();
}
fis.close();
out.close();
}
}
catch(Exception e)
{
e.printStackTrace();
}
}
}

```

### **ftpclient:**

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class ftpclient extends JFrame implements ActionListener
{
JButton b1,b2;
JLabel l1,l2,msg1,msg2;
JPanel p1,p2,p3;
public ftpclient()
{
b1=new JButton("upload");
b2=new JButton("Download");
l1=new JLabel("Upload a file");

```

```

l2=new JLabel("Download a file");
msg1=new JLabel("");
msg2=new JLabel("");
p1=new JPanel();
p2=new JPanel();
p3=new JPanel();
p1.add(l1);
p1.add(b1);
p3.add(msg1);
p2.add(l2);
p2.add(b2);
p3.add(msg2);
b1.addActionListener(this);
b2.addActionListener(this);
add("North",p1);
add("Center",p3);
add("South",p2);
setVisible(true);
setSize(300,300);
}
public void actionPerformed(ActionEvent ae)
{
try
{
if(b1.getModel().isArmed())
{
Socket s=new Socket(InetAddress.getLocalHost(),8020);
System.out.println("CLIENT CONNECTED TO SERVER");
JFileChooser j=new JFileChooser();
int val;
val=j.showOpenDialog(ftpclient.this);
String filename=j.getSelectedFile().getName();
String path=j.getSelectedFile().getPath();
PrintStream out=new PrintStream(s.getOutputStream());
out.println("Upload");
out.println(filename);
FileInputStream fis=new FileInputStream(path);
int n=fis.read();
while(n!=-1)
{

```

```

out.print((char)n);
n=fis.read();
}
fis.close();
out.close();
msg1.setText(filename + "is successfully uploaded to ");
repaint();
}

if(b2.getModel().isArmed())
{
Socket s=new Socket(InetAddress.getLocalHost(),8020);
System.out.println("CLIENT CONNECTED TO SERVER");
String remoteadd=s.getRemoteSocketAddress().toString();
System.out.println(remoteadd);
JFileChooser j1=new JFileChooser(remoteadd);
int val;
val=j1.showOpenDialog(ftpclient.this);
String filename=j1.getSelectedFile().getName();
String filepath=j1.getSelectedFile().getPath();
PrintStream out=new PrintStream(s.getOutputStream());
out.println("DOWNLOADING....");
out.println(filepath);
FileOutputStream fout=new FileOutputStream(filename);
DataInputStream fromserver=new DataInputStream(s.getInputStream());
int ch;
while((ch=fromserver.read())!=-1)
{
fout.write((char)ch);
}
fout.close();
msg2.setText(filename + "IS DOWNLOADED");
repaint();
}
}
catch(Exception e)
{
System.out.println(e);
}
}

```

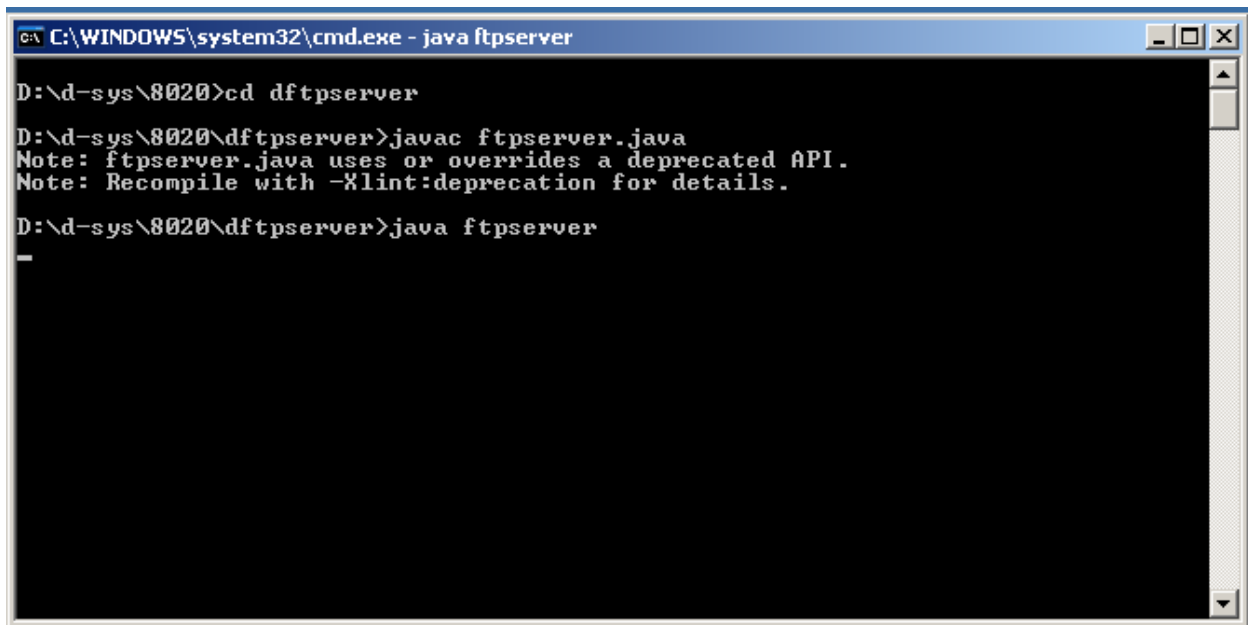


```
public static void main(String args[])
{
    new ftpclient();
}
}
```

### EXECUTION STEPS:

1. Create a folder and write server program.
2. Create another folder and write client program.
3. After compilation the UPLOADED file will be stored in client's folder and DOWNLOADED file in server's folder.

### OUTPUT:



```
C:\WINDOWS\system32\cmd.exe - java ftpserver

D:\d-sys\8020>cd dftpserver

D:\d-sys\8020\dftpserver>javac ftpserver.java
Note: ftpserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\d-sys\8020\dftpserver>java ftpserver
-
```

```

C:\WINDOWS\system32\cmd.exe - java ftpclient
D:\d-sys\8020>cd dftpclient
D:\d-sys\8020\dftpclient>javac ftpclient.java
D:\d-sys\8020\dftpclient>java ftpclient
_

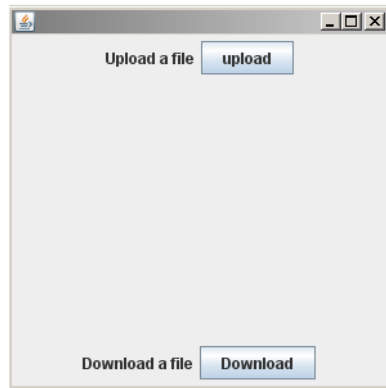
```

```

C:\WINDOWS\system32\cmd.exe
D:\d-sys\8020>cd dftpserver
D:\d-sys\8020\dftpserver>javac ftpserver.java
Note: ftpserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\d-sys\8020\dftpserver>java ftpserver
SERVER SOCKET IS CREATED
Upload text
D:\d-sys\8020\dftpserver>

C:\WINDOWS\system32\cmd.exe - java ftpclient
D:\d-sys\8020>cd dftpclient
D:\d-sys\8020\dftpclient>javac ftpclient.java
D:\d-sys\8020\dftpclient>java ftpclient
CLIENT CONNECTED TO SERVER

```



```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dcet>d:
D:\>cd d-sys
D:\d-sys>cd 8020
D:\d-sys\8020>cd dftpserver
D:\d-sys\8020\dftpserver>javac ftpserver.java
Note: ftpserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\d-sys\8020\dftpserver>java ftpserver
SERVER SOCKET IS CREATED
D:\d-sys\8020\dftpserver>

C:\WINDOWS\system32\cmd.exe - java ftpclient
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dcet>d:
D:\>cd 8020
The system cannot find the path specified.
D:\>cd d-sys
D:\d-sys>cd 8020
D:\d-sys\8020>cd dftpclient.java
The system cannot find the path specified.
D:\d-sys\8020>cd dftpclient
D:\d-sys\8020\dftpclient>javac ftpclient.java
D:\d-sys\8020\dftpclient>java ftpclient
java.net.ConnectException: Connection refused: connect
CLIENT CONNECTED TO SERVER
cc42/192.168.0.42:8020
java.net.SocketException: Connection reset

```

## **2. IMPLEMENT A MINI DNS PROTOCOL USING RPC.**

### **DNS USING RPC:**

The Domain Name System (DNS) is a hierarchical naming system built on a distributed database for computers, services, or any resource connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities. Most importantly, it translates domain names meaningful to humans into the numerical identifiers associated with networking equipment for the purpose of locating and addressing these devices worldwide.

An often-used analogy to explain the Domain Name System is that it serves as the phone book for the Internet by translating human-friendly computer hostnames into IP addresses. For example, the domain name `www.example.com` translates to the addresses `192.0.32.10` (IPv4) and `2620:0:2d0:200::10` (IPv6).

The Domain Name System makes it possible to assign domain names to groups of Internet resources and users in a meaningful way, independent of each entity's physical location. Because of this, World Wide Web (WWW) hyperlinks and Internet contact information can remain consistent and constant even if the current Internet routing arrangements change or the participant uses a mobile device.

Internet domain names are easier to remember than IP addresses such as `208.77.188.166` (IPv4) or `2001:db8:1f70::999:de8:7648:6e8` (IPv6). Users take advantage of this when they recite meaningful Uniform Resource Locators (URLs) and e-mail addresses without having to know how the computer actually locates them.

The Domain Name System distributes the responsibility of assigning domain names and mapping those names to IP addresses by designating authoritative name servers for each domain. Authoritative name servers are assigned to be responsible for their particular domains, and in turn can assign other authoritative name servers for their sub-domains. This mechanism has made the DNS distributed and fault tolerant and has helped avoid the need for a single central register to be continually consulted and updated.

In general, the Domain Name System also stores other types of information, such as the list of mail servers that accept email for a given Internet domain. By

providing a worldwide, distributed keyword-based redirection service, the Domain Name System is an essential component of the functionality of the Internet.

In computer science, a remote procedure call (RPC) is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote. When the software in question uses object-oriented principles, RPC is called remote invocation or remote method invocation.

An RPC is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters. The remote server sends a response to the client, and the application continues its process. There are many variations and subtleties in various implementations, resulting in a variety of different (incompatible) RPC protocols. While the server is processing the call, the client is blocked (it waits until the server has finished processing before resuming execution).

An important difference between remote procedure calls and local calls is that remote calls can fail because of unpredictable network problems. Also, callers generally must deal with such failures without knowing whether the remote procedure was actually invoked. Idempotent procedures (those that have no additional effects if called more than once) are easily handled, but enough difficulties remain that code to call remote procedures is often confined to carefully written low-level subsystems.

### **Sequence of events during a RPC:**

1. The client calls the Client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
2. The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshalling.
3. The kernel sends the message from the client machine to the server machine.
4. The kernel passes the incoming packets to the server stub.
5. Finally, the server stub calls the server procedure. The reply traces the same in other direction.

**dnsserver:**

```
import java.util.Properties;
import java.net.*;
import java.awt.*;
import java.io.*;
class dnsserver
{
    ServerSocket server;
    Socket connection;
    DataInputStream input;
    DataOutputStream output;
    FileInputStream fin=null;
    FileOutputStream fout=null;
    Properties clients;
    dnsserver()
    {
        try
        {
            clients=new Properties();
            fin=new FileInputStream("Namelist.dat");
            if(fin!=null)
            {
                clients.load(fin);
                fin.close();
            }
            server=new ServerSocket(1500);
            connection=server.accept();
            output=new DataOutputStream(connection.getOutputStream());
            input=new DataInputStream(connection.getInputStream());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    void runserver()
    {
        while(true)
```

```
{
try
{
String s=input.readUTF();

if(s.equals("lookup"))
{
String s2=input.readUTF();
System.out.println(s2);
String s1=lookup(s2);
if(s1==null)
output.writeUTF("Host name not found");
else
output.writeUTF("IP address of "+s2+"is"+s1);
}

if(s.equals("add"))
{
String s1=input.readUTF();
String s2=input.readUTF();
boolean b=addHost(s1,s2);
if(b==true)
output.writeUTF("Host name registered");
else
output.writeUTF("Ip address already exists");
}

if(s.equals("remove"))
{
String s1=input.readUTF();
boolean b=removeHost(s1);
if(b==true)
output.writeUTF("Hostname removed");
else
output.writeUTF("Invalid hostname");
}
}
catch(Exception e)
{
e.printStackTrace();
}
```

```
}  
}  
}
```

```
boolean addHost(String name,String ip)  
{  
if(clients.get(name)!=null)  
return false;  
else  
clients.put(name,ip);  
try  
{  
fout=new FileOutputStream("NameList.dat");  
clients.store(fout,"Namespace");  
fout.close();  
}  
catch(IOException ex)  
{  
ex.printStackTrace();  
}  
return true;  
}
```

```
boolean removeHost(String name)  
{  
String client=(String)clients.get(name);  
if(client!=null)  
clients.remove(name);  
try  
{  
fout=new FileOutputStream("NameList.dat");  
clients.store(fout,"NameSpace");  
fout.close();  
}  
catch(IOException ex)  
{  
ex.printStackTrace();  
}  
return true;  
}
```



```
String lookup(String host)
{
String ip=(String)clients.get(host);
return ip;
}
```

```
public static void main(String args[])
{
dnsserver ds=new dnsserver();
ds.runserver();
}
}
```

### **dnsclient:**

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
public class dnsclient extends Frame implements ActionListener
{
Button b1,b2,b3,b4;
Panel p1,p2,p3;
Label l1,l2;
TextField t1,t2,t3;
Socket s1;
String s;
DataOutputStream output;
DataInputStream input;

dnsclient()
{
super("dnsclient");
b1=new Button("add");
b2=new Button("lookup");
b3=new Button("remove");
b4=new Button("exit");
p1=new Panel();
p2=new Panel();
```

```

p3=new Panel();
l1=new Label("Host name:");
l2=new Label("IP address");
t1=new TextField(" ",25);
t2=new TextField(" ",25);
t3=new TextField(" ",25);
p1.add(l1);
p1.add(t1);
p1.add(l2);
p1.add(t2);
p2.add(b1);
p2.add(b2);
p2.add(b3);
p2.add(b4);
p3.add(t3);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
add("South",p1);
add("Center",p3);
add("North",p2);
setVisible(true);
setSize(300,300);
try
{
Socket s1=new Socket(InetAddress.getLocalHost(),1500);
if(s1!=null)
System.out.println("Connected to server");
output=new DataOutputStream(s1.getOutputStream());
input=new DataInputStream(s1.getInputStream());
}
catch(Exception e)
{
e.printStackTrace();
}
}

public void actionPerformed(ActionEvent ae)
{

```

```
try{
if(ae.getSource()==b1)
{
output.writeUTF("add");
output.writeUTF(t1.getText());
output.writeUTF(t2.getText());
s=input.readUTF();
t3.setText(s);
}

if(ae.getSource()==b2)
{
output.writeUTF("lookup");
output.writeUTF(t1.getText());
s=input.readUTF();
t3.setText(s);
}

if(ae.getSource()==b3)
{
output.writeUTF("remove");
output.writeUTF(t1.getText());
s=input.readUTF();
t3.setText(s);
}

if(ae.getSource()==b4)
{
System.exit(1);
}
}
catch(Exception ex)
{
ex.printStackTrace();
}
}

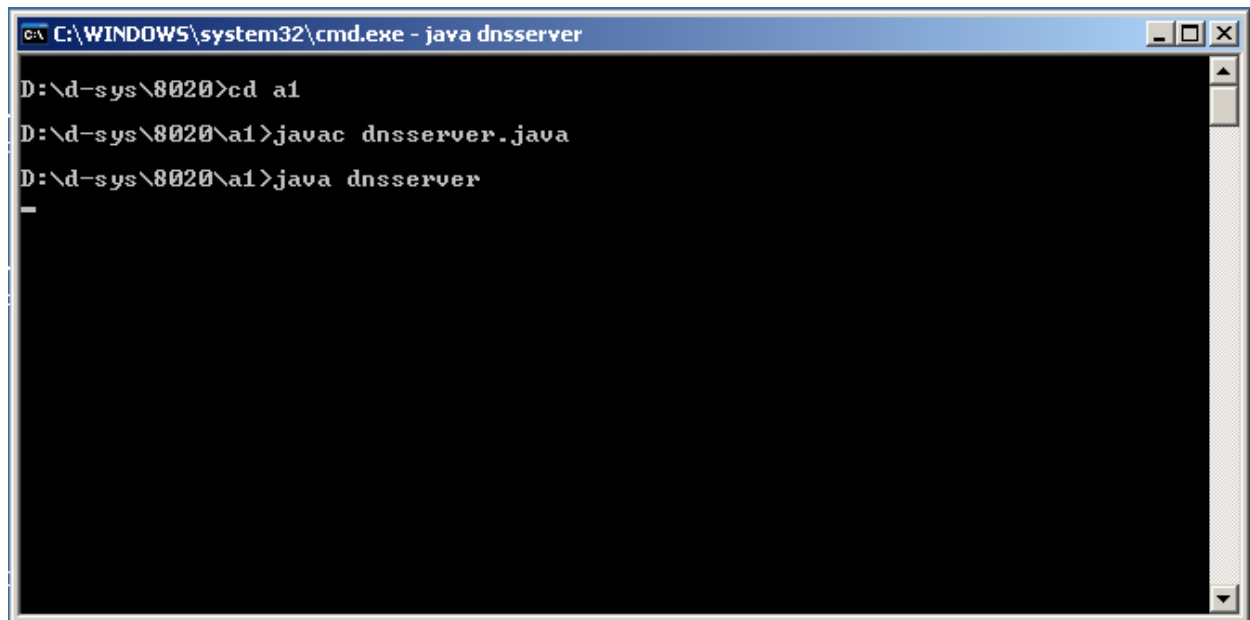
public static void main(String args[])
{
dnsclient dc=new dnsclient();
```

```
}  
}
```

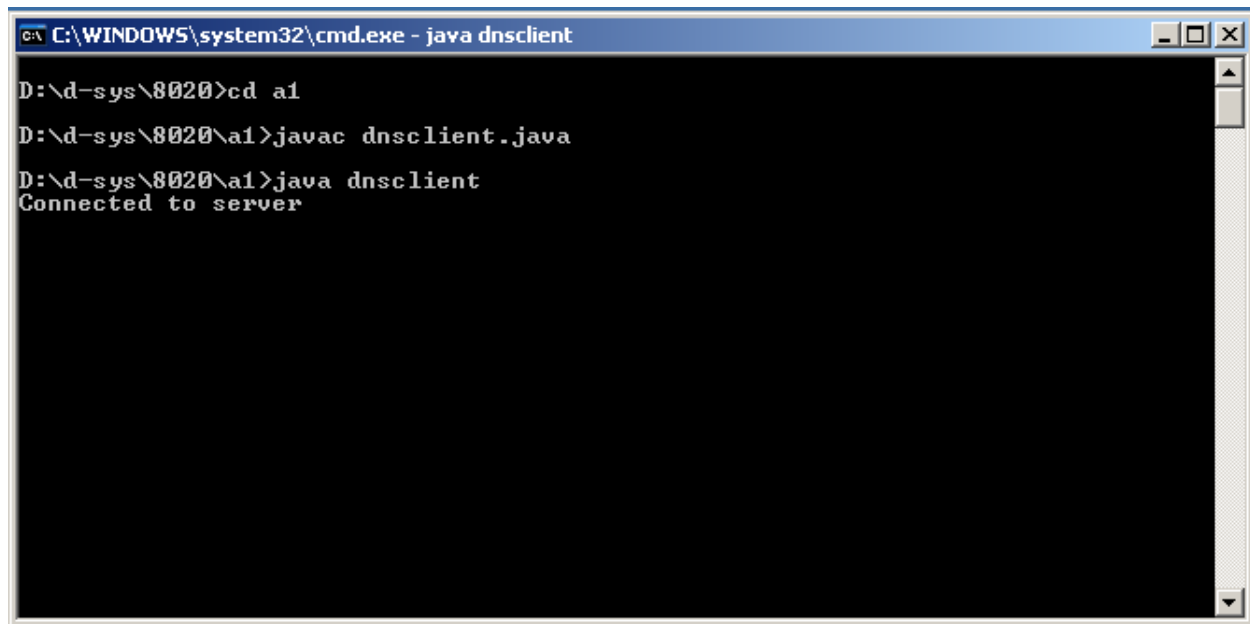
### EXECUTION STEPS:

1. Create a separate folder and write both server and client programs.
2. Create an empty NameList.dat file.
3. During execution, after adding host name and IP address we can see the added information in NameList.dat.

### OUTPUT:

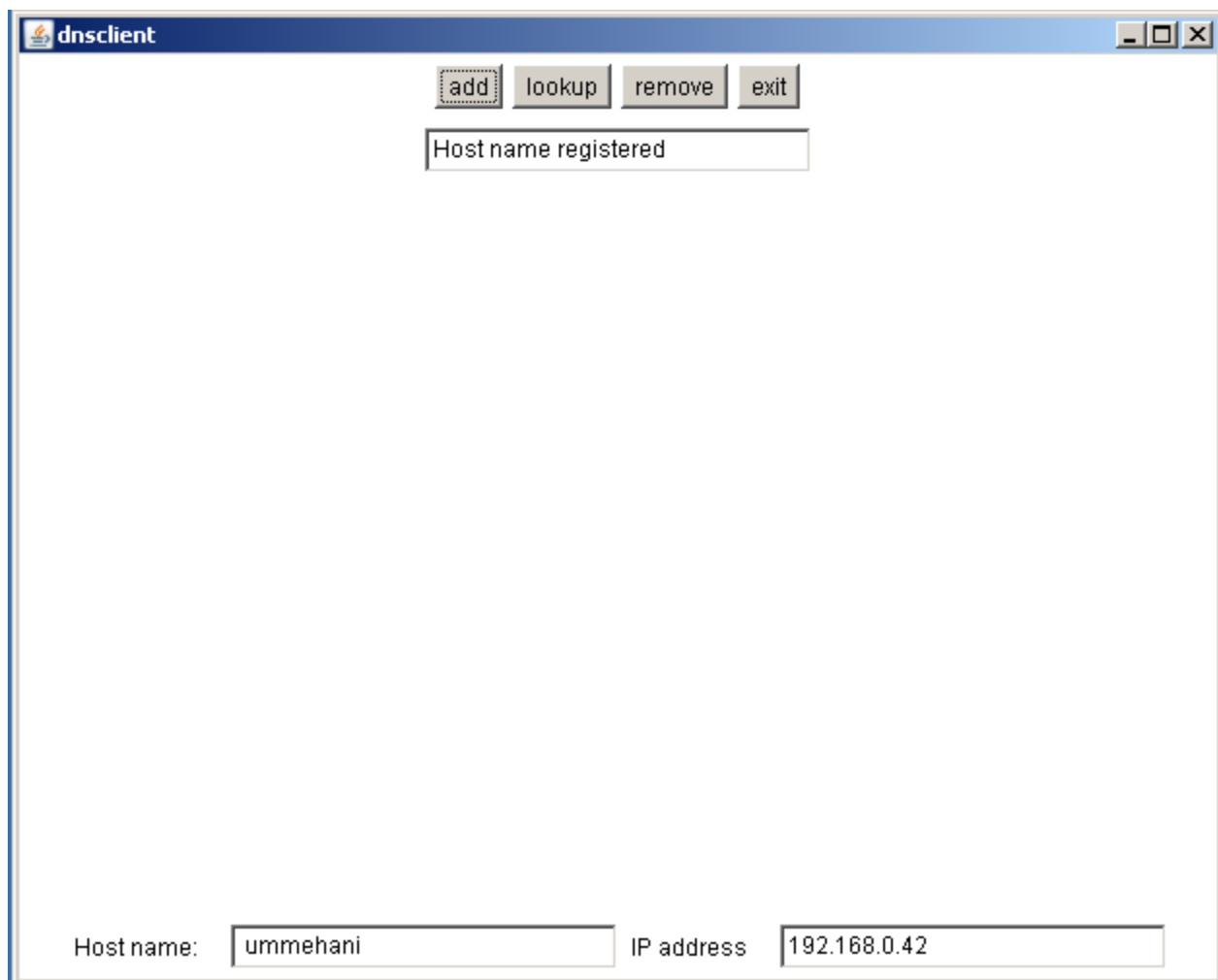


```
C:\WINDOWS\system32\cmd.exe - java dnsserver  
D:\d-sys\8020>cd a1  
D:\d-sys\8020\1>javac dnsserver.java  
D:\d-sys\8020\1>java dnsserver  
_
```



```
C:\WINDOWS\system32\cmd.exe - java dnsclient

D:\d-sys\8020>cd a1
D:\d-sys\8020\>javac dnsclient.java
D:\d-sys\8020\>java dnsclient
Connected to server
```

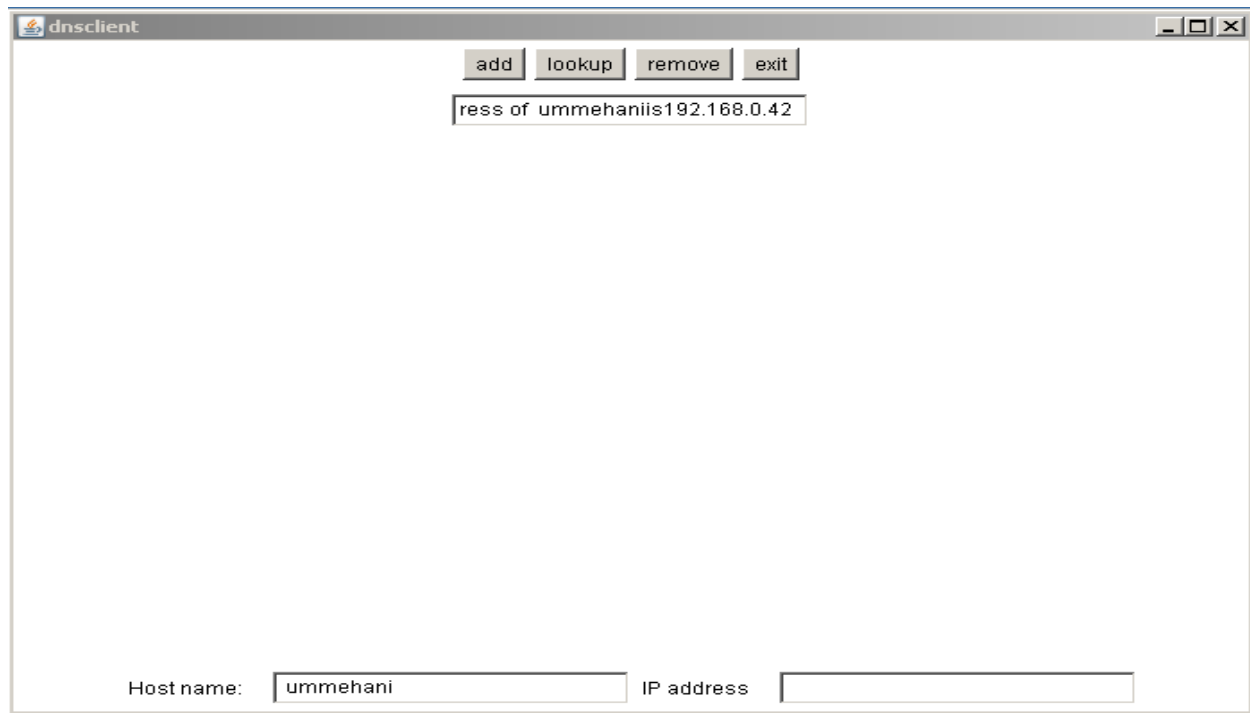


dnsclient

add lookup remove exit

Host name registered

Host name: ummehani IP address 192.168.0.42



The screenshot shows a window titled "dnsclient" with a menu bar containing "add", "lookup", "remove", and "exit". The "add" button is highlighted. Below the menu bar, a text box contains the text "ress of ummehaniis192.168.0.42". At the bottom of the window, there are two input fields: "Host name:" with the value "ummehani" and "IP address:" which is empty.



The screenshot shows the same "dnsclient" window. The "remove" button is now highlighted. The text box below the menu bar contains the text "Hostname removed". The "Host name:" and "IP address:" input fields at the bottom are now empty.

### 3. IMPLEMENT A CHAT SERVER USING JAVA.

#### CHATSERVER :

Chat server is a standalone application that is made up the combination of two-application, server application (which runs on server side) and client application (which runs on client side). This application is using for chatting in LAN. To start chatting you must be connected with the server after that your message can broadcast to each and every client.

For making this application we are using some core java features like swing, collection, networking, I/O Streams and threading also. In this application we have one server and any number of clients (which are to be communicated with each other). For making 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. For running whole client operation we can run the Login.java.

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.

#### The Server side application follows these steps:

- \* Firstly creates a new server socket by the ServerSocket ss = new ServerSocket(1004);

- \* After creating the ServerSocket it accepts the client socket and add this socket into arraylist.

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

- \* After getting the client socket it creates a thread and make DataInputStream for this socket. After creating the input stream its read the user name and add it to arraylist and this arraylist object write 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);
}

```

\* After this it makes a new thread and makes one DataInputStream for reading the messages which sends by the client and after reading the message its 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();
}

```

\* If any client Logged out then server received the client name and server remove it from the arraylist. And sends this updated arraylist to all client.

```

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

```

### **Client Side Application:**

For creating the Client side application firstly creates the login frame it consist one textfield and the login button. After hitting the login button it shows the next frame that Client Frame and it consist one textfield for writing the message and one send button for sending it. And two list boxes, one is for showing the all messages and another list box is use to show the all user names. This frame has one more button that is Logout button for terminating the chat.

The Client side application follows these steps :

\* In the client sides firstly creates a new socket and specifies the address and port of the server and establish the connection with the Server.

```

Socket s=new Socket("localhost",1004);

```



Note - Instead of "localhost" you have to write the server IP address or computer name.

\* After that client makes a new thread and DataInputStream, ObjectInputStream and DataOutputStream for sending the user name and retrieving the list of all users and add the all user name in its list box through iterator.

```
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF(name);
DataInputStream din1=new DataInputStream(s.getInputStream());
ObjectInputStream obj=new ObjectInputStream(din1);
ArrayList alname=new ArrayList();
Alname=(ArrayList)obj.readObject();
String lname;
Iterator i1=alname.iterator();
while(i1.hasNext()){
lname=(String)i1.next();
model1.addElement(lname);
}
```

Note – model1 is the object of DefaultListModel that is used to add the element in JList Box.

\* Now we make one new thread for sending and receiving the messages from the server. It does this task by using DataInputStream and DataOutputStream.

```
dout.writeUTF(str); // for sending the messages
str1=din.readUTF(); // receiving the messages
model.addElement(str1) // add these messages to JList Box
```

\* When the client is logged out it sends its name and message “User\_Name has Logged out” and terminate the chatting

```
dout.writeUTF(name+" has Logged out");
dout.writeUTF(name);
```

## ServerChat

```

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class ServerChat extends JFrame implements ActionListener
{
    JButton send = new JButton("SEND");
    JTextArea area = new JTextArea(20,20);
    JScrollPane jsp = new JScrollPane(area);
    JTextField text = new JTextField(20);
    JLabel label = new JLabel("Enter Ur Text :: ");
    JPanel panel = new JPanel();
    ServerSocket server;
    Socket client;
    DataInputStream din;
    PrintWriter pw;
    public ServerChat()
    {
        super("Server Window");
        panel.add(label);
        panel.add(text);
        panel.add(send);
        panel.setBorder(new LineBorder(Color.red,2,true));
        add(panel,BorderLayout.SOUTH);
        jsp.setBorder(new EmptyBorder(10,10,10,10));
        add(jsp);
        area.setFont(new Font("TimesRoman",Font.PLAIN,15));
        area.setEditable(false);
    }
}

```

```
send.addActionListener(this);
text.addActionListener(this);
setDefaultCloseOperation(EXIT_ON_CLOSE);
setSize(500,500);
setLocationRelativeTo(null);
setVisible(true);
try
{
    server = new ServerSocket(7000);
    area.setText("Server Is Waiting For Client");
    client = server.accept();
    area.append("\nClient Is Now Connected");
    pw = new PrintWriter(client.getOutputStream());
    din = new DataInputStream(client.getInputStream());
}
catch(Exception e)
{
    System.out.println("\n\tException " + e);
}
}
public void receive()
{
    try
    {
        String s;
        while((s=din.readLine())!=null)
        {
            String temp = "\nClient : " + s ;
            area.append(temp);
        }
    }
    catch(Exception e)
    {
        System.out.println("Exception " + e);
    }
}
```

```

    }
    public void actionPerformed(ActionEvent ae)
    {
        try
        {
            String s = "\nServer : " + text.getText() ;
            area.append(s);
            pw.println(text.getText());
            pw.flush();
            text.setText("");
        }
        catch(Exception e)
        {
            System.out.println("Exception " + e);
        }
    }
    public static void main(String args[])
    {
        new ServerChat().receive();
    }
}

```

### Client Chat

```

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class ClientChat extends JFrame implements ActionListener
{

```

```

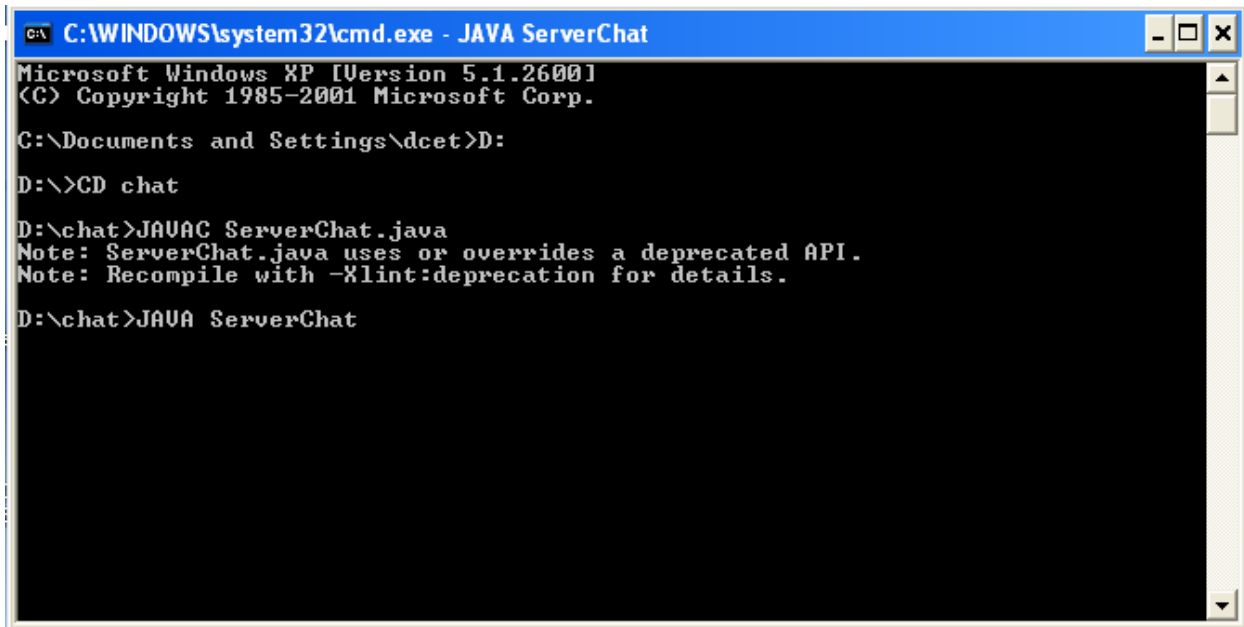
JButton send = new JButton("SEND");
JTextArea area = new JTextArea(20,20);
JScrollPane jsp = new JScrollPane(area);
JTextField text = new JTextField(20);
JLabel label = new JLabel("Enter Ur Text :: ");
JPanel panel = new JPanel();
Socket client;
DataInputStream din;
PrintWriter pw;
public ClientChat()
{
    super("Client Window");
    panel.add(label);
    panel.add(text);
    panel.add(send);
    panel.setBorder(new LineBorder(Color.red,2,true));
    add(panel,BorderLayout.SOUTH);
    jsp.setBorder(new EmptyBorder(10,10,10,10));
    add(jsp);
    send.addActionListener(this);
    text.addActionListener(this);
    area.setFont(new Font("TimesRoman",Font.PLAIN,15));
    area.setEditable(false);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(500,500);
    setLocationRelativeTo(null);
    setVisible(true);
    try
    {
        client = new Socket("localhost",7000);
        pw = new PrintWriter(client.getOutputStream());
        din = new DataInputStream(client.getInputStream());
    }
    catch(Exception e)
    {

```

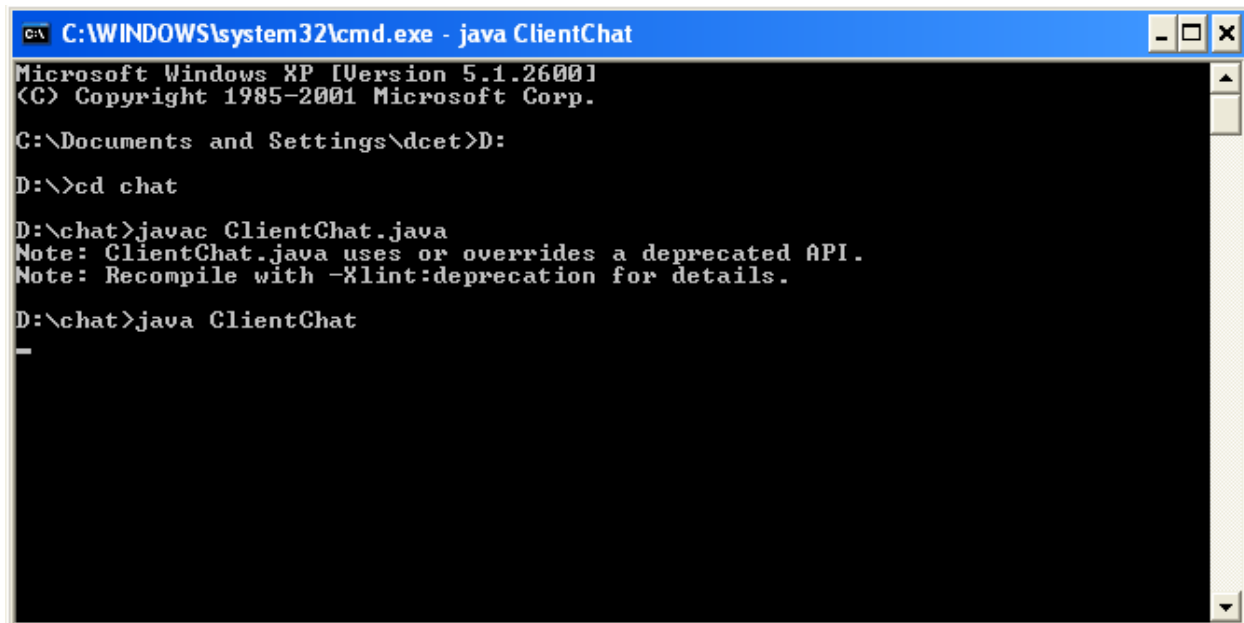
```

        System.out.println("\n\tException " + e);
    }
}
public void receive()
{
    try
    {
        String s;
        while((s=din.readLine())!=null)
        {
            String temp = "\nServer : " + s ;
            area.append(temp);
        }
    }
    catch(Exception e)
    {
        System.out.println("Exception " + e);
    }
}
public void actionPerformed(ActionEvent ae)
{
    try
    {
        String s = "\nClient : " + text.getText() ;
        area.append(s);
        pw.println(text.getText());
        pw.flush();
        text.setText("");
    }
    catch(Exception e)
    {
        System.out.println("Exception " + e);
    }
}
public static void main(String args[])
{
    new ClientChat().receive();
}
}

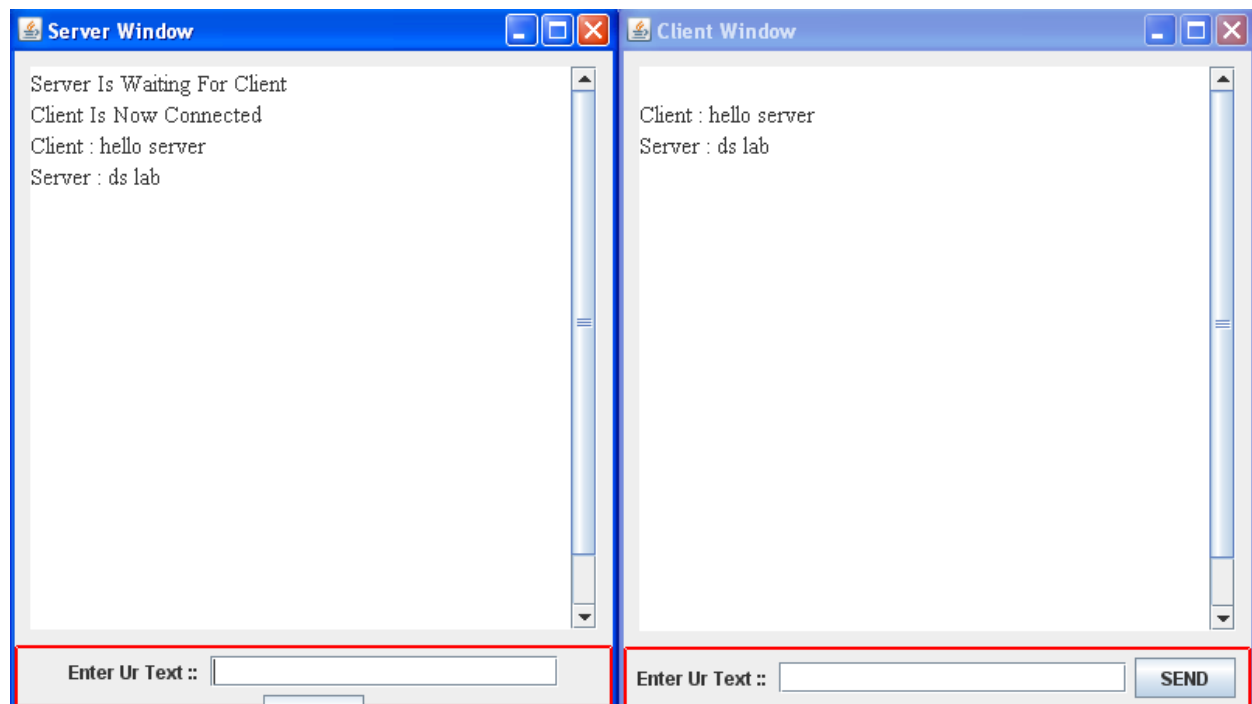
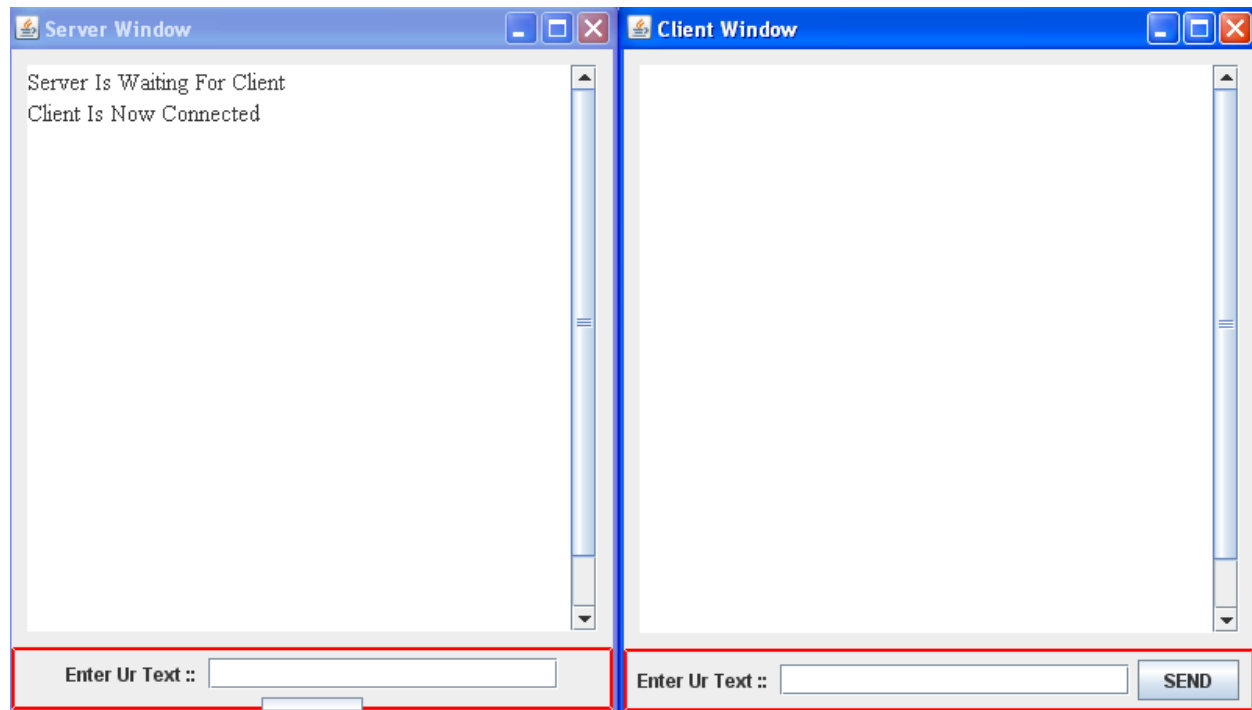
```

Out put:

```
C:\WINDOWS\system32\cmd.exe - JAVA ServerChat
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\dcet>D:
D:\>CD chat
D:\chat>JAVAC ServerChat.java
Note: ServerChat.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\chat>JAVA ServerChat
```



```
C:\WINDOWS\system32\cmd.exe - java ClientChat
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\dcet>D:
D:\>cd chat
D:\chat>javac ClientChat.java
Note: ClientChat.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\chat>java ClientChat
```





## 4. IMPLEMENTATION OF JAVA API ILLUSTRATING IPC USING TCP PROTOCOL.

The java interface to tcp streams is provided In the classes ServerSocket and Socket

### ServerSocket:

This class is intended for use by a server to create a socket at a server port for listening for connect request from the clients. Its accept method gets a connect request from the queue or if the queue is empty it blocks until one arrives. The result of executing accept is an instance of Socket- a socket for giving access to streams for communicating within the client.

### Socket:

This class is for use by a pair of processes within a connection the client uses a constructor to create a socket , specifying the DNS host name and port of a server. This constructor not only creates a socket associated with a local port also connects it to the specified remote computer and port number. It can throw an UnknownHostException the host name is wrong or an IOException if an IO error occurs.

The Socket class provides methods getInputStream and getOutputStream for accessing the two streams associated with a socket. The return types of this methods are InputStream and OutputStream respectively- abstract classes that defines methods for reading and writing bytes . The return values can be used as the argument of constructor for suitable input and output streams. DataInputStream and DataOutputStream which allows binary representation of primitive data types to be read and written in machine independent manner.

The client end server process use the method writeUTF of DataOutputStream to write it to the output Stream and the method readUTF of DataInputStream to read it from the Input Stream . UTF-8 is an encoding that represents strings in a particular format

Class ServerSocket  
[java.lang.Object](#)  
 └ **java.net.ServerSocket**

Direct Known Subclasses:  
[SSLServerSocket](#)

---

public class **ServerSocket**

extends [Object](#)

This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

The actual work of the server socket is performed by an instance of the `SocketImpl` class. An application can change the socket factory that creates the socket implementation to configure itself to create sockets appropriate to the local firewall.

**Since:**

JDK1.0

### Constructor Summary

[ServerSocket](#)()

Creates an unbound server socket.

[ServerSocket](#)(int port)

Creates a server socket, bound to the specified port.

[ServerSocket](#)(int port, int backlog)

Creates a server socket and binds it to the specified local port number, with the specified backlog.

[ServerSocket](#)(int port, int backlog, [InetAddress](#) bindAddr)

Create a server with the specified port, listen backlog, and local IP address to bind to.

### Method Summary

[Socket](#)

[accept](#)()

Listens for a connection to be made to this socket and

	accepts it.
void	<a href="#"><b>bind</b></a> ( <a href="#">SocketAddress</a> endpoint) Binds the ServerSocket to a specific address (IP address and port number).
void	<a href="#"><b>bind</b></a> ( <a href="#">SocketAddress</a> endpoint, int backlog) Binds the ServerSocket to a specific address (IP address and port number).
void	<a href="#"><b>close</b></a> () Closes this socket.
<a href="#">ServerSocketChannel</a>	<a href="#"><b>getChannel</b></a> () Returns the unique <a href="#">ServerSocketChannel</a> object associated with this socket, if any.
<a href="#">InetAddress</a>	<a href="#"><b>getInetAddress</b></a> () Returns the local address of this server socket.
int	<a href="#"><b>getLocalPort</b></a> () Returns the port on which this socket is listening.
<a href="#">SocketAddress</a>	<a href="#"><b>getLocalSocketAddress</b></a> () Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
int	<a href="#"><b>getReceiveBufferSize</b></a> () Gets the value of the SO_RCVBUF option for this ServerSocket, that is the proposed buffer size that will be used for Sockets accepted from this ServerSocket.
boolean	<a href="#"><b>getReuseAddress</b></a> () Tests if SO_REUSEADDR is enabled.
int	<a href="#"><b>getSoTimeout</b></a> () Retrieve setting for SO_TIMEOUT.
protected void	<a href="#"><b>implAccept</b></a> ( <a href="#">Socket</a> s) Subclasses of ServerSocket use this method to override accept() to return their own subclass of socket.
boolean	<a href="#"><b>isBound</b></a> () Returns the binding state of the ServerSocket.
boolean	<a href="#"><b>isClosed</b></a> () Returns the closed state of the ServerSocket.

void	<a href="#"><u>setReceiveBufferSize</u></a> (int size) Sets a default proposed value for the SO_RCVBUF option for sockets accepted from this ServerSocket.
void	<a href="#"><u>setReuseAddress</u></a> (boolean on) Enable/disable the SO_REUSEADDR socket option.
static void	<a href="#"><u>setSocketFactory</u></a> ( <a href="#"><u>SocketImplFactory</u></a> fac) Sets the server socket implementation factory for the application.
void	<a href="#"><u>setSoTimeout</u></a> (int timeout) Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
<a href="#"><u>String</u></a>	<a href="#"><u>toString</u></a> () Returns the implementation address and implementation port of this socket as a String.

#### Methods inherited from class java.lang.[Object](#)

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [wait](#), [wait](#), [wait](#)

### ServerSocket

public **ServerSocket**()

throws [IOException](#)

Creates an unbound server socket.

#### Throws:

[IOException](#) - IO error when opening the socket.

### ServerSocket

public **ServerSocket**(int port)

throws [IOException](#)

Creates a server socket, bound to the specified port. A port of 0 creates a socket on any free port.

The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused.

If the application has specified a server socket factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its `checkListen` method is called with the port argument as its argument to ensure the operation is allowed. This could result in a `SecurityException`.

**Parameters:**

port - the port number, or 0 to use any free port.

**Throws:**

[IOException](#) - if an I/O error occurs when opening the socket.

[SecurityException](#) - if a security manager exists and its `checkListen` method doesn't allow the operation.

**See Also:**

[SocketImpl](#), [SocketImplFactory.createSocketImpl\(\)](#), [setSocketFactory\(java.net.SocketImplFactory\)](#), [SecurityManager.checkListen\(int\)](#)

---

## ServerSocket

```
public ServerSocket(int port,
                    int backlog)
    throws IOException
```

Creates a server socket and binds it to the specified local port number, with the specified backlog. A port number of 0 creates a socket on any free port.

The maximum queue length for incoming connection indications (a request to connect) is set to the backlog parameter. If a connection indication arrives when the queue is full, the connection is refused.

If the application has specified a server socket factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its `checkListen` method is called with the port argument as its argument to ensure the operation is allowed. This could result in a `SecurityException`.

The backlog argument must be a positive value greater than 0. If the value passed is equal or less than 0, then the default value will be assumed.

### Parameters:

port - the specified port, or 0 to use any free port.  
backlog - the maximum length of the queue.

### Throws:

[IOException](#) - if an I/O error occurs when opening the socket.  
[SecurityException](#) - if a security manager exists and its `checkListen` method doesn't allow the operation.

### See Also:

[SocketImpl](#), [SocketImplFactory.createSocketImpl\(\)](#), [setSocketFactory\(java.net.SocketImplFactory\)](#), [SecurityManager.checkListen\(int\)](#)

---

## ServerSocket

```
public ServerSocket(int port,  
                    int backlog,  
                    InetAddress bindAddr)  
    throws IOException
```

Create a server with the specified port, listen backlog, and local IP address to bind to. The *bindAddr* argument can be used on a multi-homed host for a `ServerSocket` that will only accept connect requests to one of its addresses. If *bindAddr* is null, it will default accepting connections on any/all local addresses. The port must be between 0 and 65535, inclusive.

If there is a security manager, this method calls its `checkListen` method with the port argument as its argument to ensure the operation is allowed. This could result in a `SecurityException`.

The backlog argument must be a positive value greater than 0. If the value passed is equal or less than 0, then the default value will be assumed.

**Parameters:**

port - the local TCP port  
backlog - the listen backlog  
bindAddr - the local InetAddress the server will bind to

**Throws:**

[SecurityException](#) - if a security manager exists and its checkListen method doesn't allow the operation.

[IOException](#) - if an I/O error occurs when opening the socket.

**Constructor Summary**

	<a href="#">Socket</a> () Creates an unconnected socket, with the system-default type of SocketImpl.
	<a href="#">Socket</a> ( <a href="#">InetAddress</a> address, int port) Creates a stream socket and connects it to the specified port number at the specified IP address.
	<a href="#">Socket</a> ( <a href="#">InetAddress</a> host, int port, boolean stream) <b>Deprecated.</b> <i>Use DatagramSocket instead for UDP transport.</i>
	<a href="#">Socket</a> ( <a href="#">InetAddress</a> address, int port, <a href="#">InetAddress</a> localAddr, int localPort) Creates a socket and connects it to the specified remote address on the specified remote port.
protected	<a href="#">Socket</a> ( <a href="#">SocketImpl</a> impl) Creates an unconnected Socket with a user-specified SocketImpl.
	<a href="#">Socket</a> ( <a href="#">String</a> host, int port) Creates a stream socket and connects it to the specified port number on the named host.
	<a href="#">Socket</a> ( <a href="#">String</a> host, int port, boolean stream) <b>Deprecated.</b> <i>Use DatagramSocket instead for UDP transport.</i>
	<a href="#">Socket</a> ( <a href="#">String</a> host, int port, <a href="#">InetAddress</a> localAddr, int localPort) Creates a socket and connects it to the specified remote host on the specified remote port.

<b>Method Summary</b>	
void	<b><u>bind</u></b> ( <a href="#">SocketAddress</a> bindpoint) Binds the socket to a local address.
void	<b><u>close</u></b> () Closes this socket.
void	<b><u>connect</u></b> ( <a href="#">SocketAddress</a> endpoint) Connects this socket to the server.
void	<b><u>connect</u></b> ( <a href="#">SocketAddress</a> endpoint, int timeout) Connects this socket to the server with a specified timeout value.
<a href="#">SocketChannel</a>	<b><u>getChannel</u></b> () Returns the unique <a href="#">SocketChannel</a> object associated with this socket, if any.
<a href="#">InetAddress</a>	<b><u>getInetAddress</u></b> () Returns the address to which the socket is connected.
<a href="#">InputStream</a>	<b><u>getInputStream</u></b> () Returns an input stream for this socket.
boolean	<b><u>getKeepAlive</u></b> () Tests if SO_KEEPALIVE is enabled.
<a href="#">InetAddress</a>	<b><u>getLocalAddress</u></b> () Gets the local address to which the socket is bound.
int	<b><u>getLocalPort</u></b> () Returns the local port to which this socket is bound.
<a href="#">SocketAddress</a>	<b><u>getLocalSocketAddress</u></b> () Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
boolean	<b><u>getOOBInline</u></b> () Tests if OOBINLINE is enabled.
<a href="#">OutputStream</a>	<b><u>getOutputStream</u></b> () Returns an output stream for this socket.
int	<b><u>getPort</u></b> () Returns the remote port to which this socket is connected.



int	<a href="#"><b>getReceiveBufferSize()</b></a> Gets the value of the SO_RCVBUF option for this Socket, that is the buffer size used by the platform for input on this Socket.
<a href="#"><u>SocketAddress</u></a>	<a href="#"><b>getRemoteSocketAddress()</b></a> Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
boolean	<a href="#"><b>getReuseAddress()</b></a> Tests if SO_REUSEADDR is enabled.
int	<a href="#"><b>getSendBufferSize()</b></a> Get value of the SO_SNDBUF option for this Socket, that is the buffer size used by the platform for output on this Socket.
int	<a href="#"><b>getSoLinger()</b></a> Returns setting for SO_LINGER.
int	<a href="#"><b>getSoTimeout()</b></a> Returns setting for SO_TIMEOUT.
boolean	<a href="#"><b>getTcpNoDelay()</b></a> Tests if TCP_NODELAY is enabled.
int	<a href="#"><b>getTrafficClass()</b></a> Gets traffic class or type-of-service in the IP header for packets sent from this Socket
boolean	<a href="#"><b>isBound()</b></a> Returns the binding state of the socket.
boolean	<a href="#"><b>isClosed()</b></a> Returns the closed state of the socket.
boolean	<a href="#"><b>isConnected()</b></a> Returns the connection state of the socket.
boolean	<a href="#"><b>isInputShutdown()</b></a> Returns whether the read-half of the socket connection is closed.
boolean	<a href="#"><b>isOutputShutdown()</b></a> Returns whether the write-half of the socket connection is closed.
void	<a href="#"><b>sendUrgentData(int data)</b></a> Send one byte of urgent data on the socket.
void	<a href="#"><b>setKeepAlive(boolean on)</b></a>

	Enable/disable SO_KEEPALIVE.
void	<a href="#"><u>setOOBInline</u></a> (boolean on) Enable/disable OOBINLINE (receipt of TCP urgent data) By default, this option is disabled and TCP urgent data received on a socket is silently discarded.
void	<a href="#"><u>setReceiveBufferSize</u></a> (int size) Sets the SO_RCVBUF option to the specified value for this Socket.
void	<a href="#"><u>setReuseAddress</u></a> (boolean on) Enable/disable the SO_REUSEADDR socket option.
void	<a href="#"><u>setSendBufferSize</u></a> (int size) Sets the SO_SNDBUF option to the specified value for this Socket.
static void	<a href="#"><u>setSocketImplFactory</u></a> ( <a href="#"><u>SocketImplFactory</u></a> fac) Sets the client socket implementation factory for the application.
void	<a href="#"><u>setSoLinger</u></a> (boolean on, int linger) Enable/disable SO_LINGER with the specified linger time in seconds.
void	<a href="#"><u>setSoTimeout</u></a> (int timeout) Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
void	<a href="#"><u>setTcpNoDelay</u></a> (boolean on) Enable/disable TCP_NODELAY (disable/enable Nagle's algorithm).
void	<a href="#"><u>setTrafficClass</u></a> (int tc) Sets traffic class or type-of-service octet in the IP header for packets sent from this Socket.
void	<a href="#"><u>shutdownInput</u></a> () Places the input stream for this socket at "end of stream".
void	<a href="#"><u>shutdownOutput</u></a> () Disables the output stream for this socket.
<a href="#"><u>String</u></a>	<a href="#"><u>toString</u></a> () Converts this socket to a String.

### Methods inherited from class `java.lang.Object`

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#),  
[wait](#), [wait](#), [wait](#)

### Constructor Detail

#### Socket

public **Socket**()

Creates an unconnected socket, with the system-default type of `SocketImpl`.

#### Since:

JDK1.1

#### Socket

protected **Socket**([SocketImpl](#) impl)

throws [SocketException](#)

Creates an unconnected `Socket` with a user-specified `SocketImpl`.

#### Parameters:

impl - an instance of a **SocketImpl** the subclass wishes to use on the `Socket`.

#### Throws:

[SocketException](#) - if there is an error in the underlying protocol, such as a TCP error.

#### Since:

JDK1.1

#### Socket

public **Socket**([String](#) host,  
int port)

throws [UnknownHostException](#),  
[IOException](#)

Creates a stream socket and connects it to the specified port number on the named host.

If the specified host is null it is the equivalent of specifying the address as [InetAddress.getByName\(null\)](#). In other words, it is equivalent to specifying an address of the loopback interface.

If the application has specified a server socket factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its `checkConnect` method is called with the host address and port as its arguments. This could result in a `SecurityException`.

**Parameters:**

host - the host name, or null for the loopback address.  
port - the port number.

**Throws:**

[UnknownHostException](#) - if the IP address of the host could not be determined.

[IOException](#) - if an I/O error occurs when creating the socket.

[SecurityException](#) - if a security manager exists and its `checkConnect` method doesn't allow the operation.

**See Also:**

[setSocketImplFactory\(java.net.SocketImplFactory\)](#), [SocketImpl](#), [SocketImplFactory.createSocketImpl\(\)](#), [SecurityManager.checkConnect\(java.lang.String, int\)](#)

## Socket

```
public Socket(InetAddress address,  
              int port)  
    throws IOException
```

Creates a stream socket and connects it to the specified port number at the specified IP address.

If the application has specified a socket factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its `checkConnect` method is called with the host address and port as its arguments. This could result in a `SecurityException`.

**Parameters:**

address - the IP address.

port - the port number.

**Throws:**

[IOException](#) - if an I/O error occurs when creating the socket.

[SecurityException](#) - if a security manager exists and its `checkConnect` method doesn't allow the operation.

**See Also:**

[setSocketImplFactory\(java.net.SocketImplFactory\)](#), [SocketImpl](#), [SocketImplFactory.createSocketImpl\(\)](#), [SecurityManager.checkConnect\(java.lang.String, int\)](#)

---

**Socket**

```
public Socket(String host,
              int port,
              InetAddress localAddr,
              int localPort)
    throws IOException
```

Creates a socket and connects it to the specified remote host on the specified remote port. The `Socket` will also `bind()` to the local address and port supplied. If the specified host is null it is the equivalent of specifying the address as [InetAddress.getByName\(null\)](#). In other words, it is equivalent to specifying an address of the loopback interface.

If there is a security manager, its `checkConnect` method is called with the host address and port as its arguments. This could result in a `SecurityException`.

**Parameters:**

host - the name of the remote host, or null for the loopback address.

port - the remote port

localAddr - the local address the socket is bound to

localPort - the local port the socket is bound to

**Throws:**

[IOException](#) - if an I/O error occurs when creating the socket.  
[SecurityException](#) - if a security manager exists and  
 its checkConnect method doesn't allow the operation.

**Since:**

JDK1.1

**See Also:**

[SecurityManager.checkConnect\(java.lang.String, int\)](#)

---

**Socket**

```
public Socket(InetAddress address,
               int port,
               InetAddress localAddr,
               int localPort)
    throws IOException
```

Creates a socket and connects it to the specified remote address on the specified remote port. The Socket will also bind() to the local address and port supplied. If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException.

**Parameters:**

address - the remote address  
 port - the remote port  
 localAddr - the local address the socket is bound to  
 localPort - the local port the socket is bound to

**Throws:**

[IOException](#) - if an I/O error occurs when creating the socket.  
[SecurityException](#) - if a security manager exists and  
 its checkConnect method doesn't allow the operation.

**Since:**

JDK1.1

**See Also:**

[SecurityManager.checkConnect\(java.lang.String, int\)](#)

---

**Socket**

```
public Socket(String host,
               int port,
               boolean stream)
    throws IOException
```

**Deprecated.** *Use DatagramSocket instead for UDP transport.*

Creates a stream socket and connects it to the specified port number on the named host.

If the specified host is null it is the equivalent of specifying the address as [InetAddress.getByName\(null\)](#). In other words, it is equivalent to specifying an address of the loopback interface.

If the stream argument is true, this creates a stream socket. If the stream argument is false, it creates a datagram socket.

If the application has specified a server socket factory, that factory's createSocketImpl method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException. If a UDP socket is used, TCP/IP related socket options will not apply.

#### Parameters:

host - the host name, or null for the loopback address.  
 port - the port number.  
 stream - a boolean indicating whether this is a stream socket or a datagram socket.

#### Throws:

[IOException](#) - if an I/O error occurs when creating the socket.  
[SecurityException](#) - if a security manager exists and its checkConnect method doesn't allow the operation.

#### See Also:

[setSocketImplFactory\(java.net.SocketImplFactory\)](#), [SocketImpl](#), [SocketImplFactory.createSocketImpl\(\)](#), [SecurityManager.checkConnect\(java.lang.String, int\)](#)

---

## Socket

```
public Socket(InetAddress host,  
              int port,  
              boolean stream)
```

throws [IOException](#)

**Deprecated.** *Use DatagramSocket instead for UDP transport.*

Creates a socket and connects it to the specified port number at the specified IP address.

If the stream argument is true, this creates a stream socket. If the stream argument is false, it creates a datagram socket.

If the application has specified a server socket factory, that factory's createSocketImpl method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its checkConnect method is called with host.getHostAddress() and port as its arguments. This could result in a SecurityException.

If UDP socket is used, TCP/IP related socket options will not apply.

### Parameters:

host - the IP address.

port - the port number.

stream - if true, create a stream socket; otherwise, create a datagram socket.

### Throws:

[IOException](#) - if an I/O error occurs when creating the socket.

[SecurityException](#) - if a security manager exists and its checkConnect method doesn't allow the operation.

### See Also:

[setSocketImplFactory\(java.net.SocketImplFactory\)](#), [SocketImpl](#), [SocketImplFactory.createSocketImpl\(\)](#), [SecurityManager.checkConnect\(java.lang.String, int\)](#)



**TCPServer:**

```
import java.io.*;
import java.net.*;
public class TCPServer
{
    public static void main(String args[])
    {
        try
        {
            int serverport=7896;
            ServerSocket listensocket=new ServerSocket(serverport);
            while(true)
            {
                Socket clientsocket=listensocket.accept();
                Connection c=new Connection(clientsocket);
            }
        }
        catch(IOException e)
        {
            System.out.println("LISTEN:"+e.getMessage());
        }
    }
}

class Connection extends Thread
{
    DataInputStream in;
    DataOutputStream out;
    Socket clientsocket;
    public Connection(Socket aclientsocket)
    {
        try
        {
            clientsocket=aclientsocket;
            in=new DataInputStream(clientsocket.getInputStream());
            out=new DataOutputStream(clientsocket.getOutputStream());
            this.start();
        }
        catch(IOException e)
        {
            
```

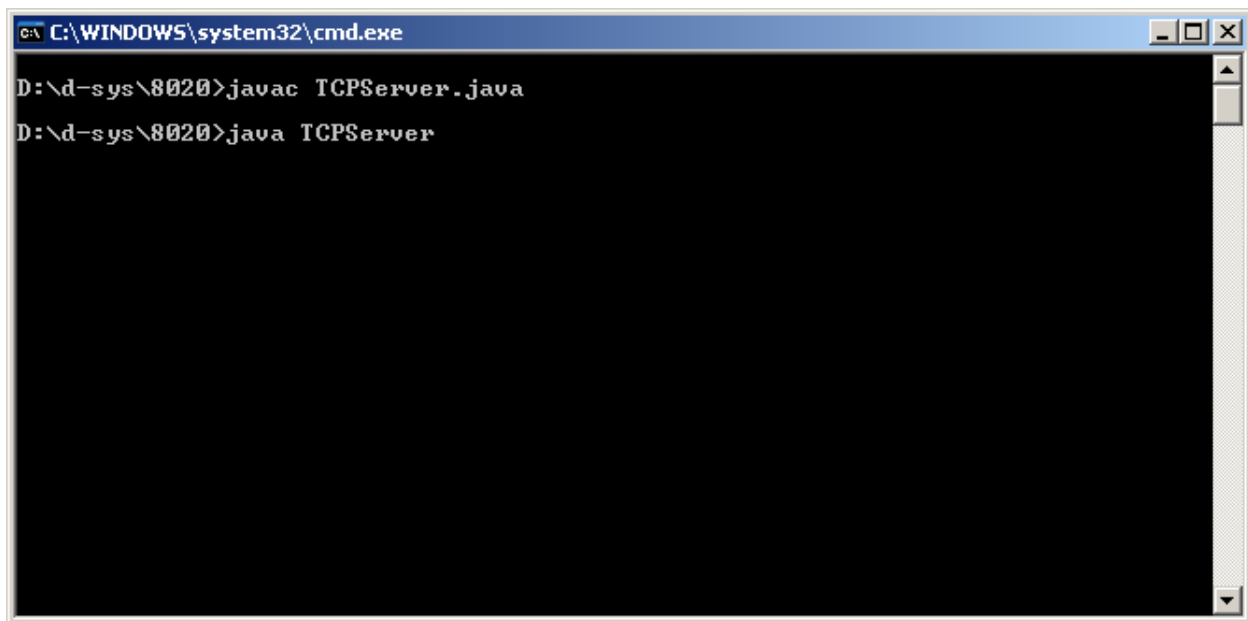
```
System.out.println("Connection:"+e.getMessage());
}
}
public void run()
{
try
{
String data=in.readUTF();
out.writeUTF(data);
}
catch(EOFException e)
{
System.out.println("EOF:"+e.getMessage());
}
catch(IOException e)
{
System.out.println("IO:"+e.getMessage());
}
finally{
try
{
clientsocket.close();
}
catch(IOException e)
{
System.out.println("close failed");
}
}
}
}
```

**TCPClient:**

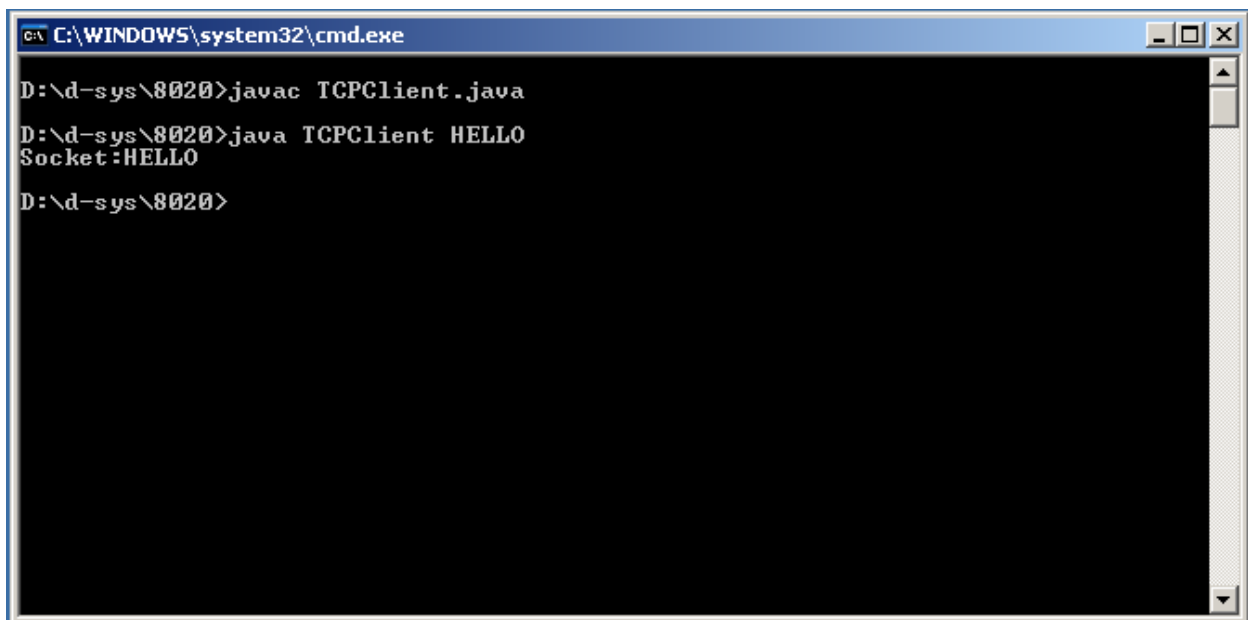
```
import java.net.*;
import java.io.*;
public class TCPClient
{
    public static void main(String args[])
    {
        Socket s=null;
        try
        {
            int serverport=7896;
            s=new Socket(args[0],serverport);
            DataInputStream in=new DataInputStream(s.getInputStream());
            DataOutputStream out=new DataOutputStream(s.getOutputStream());
            out.writeUTF(args[1]);
            String data=in.readUTF();
            System.out.println("Receive data from server:"+data);
        }
        catch(UnknownHostException e)
        {
            System.out.println("Socket:"+e.getMessage());
        }
        catch(EOFException e)
        {
            System.out.println("EOF:"+e.getMessage());
        }
        catch(IOException e)
        {
            System.out.println("IO:"+e.getMessage());
        }
        finally
        {
            if(s!=null)
            try
            {
                s.close();
            }
            catch(IOException e)
            {
            }
```

```
System.out.println("close failed");  
}  
}  
}  
}
```

## OUTPUT:



```
C:\WINDOWS\system32\cmd.exe  
D:\d-sys\8020>javac TCPServer.java  
D:\d-sys\8020>java TCPServer
```



```
C:\WINDOWS\system32\cmd.exe  
D:\d-sys\8020>javac TCPClient.java  
D:\d-sys\8020>java TCPClient HELLO  
Socket:HELLO  
D:\d-sys\8020>
```

## 5. IMPLEMENTATION OF JAVA API ILLUSTRATING IPC USING UDP PROTOCOL.

The Java API provides datagram communication by means of two classes

DatagramPacket and DatagramSocket

**DatagramPacket:** This class provides a constructor that makes an Instance out of an array bytes comprising a message, and the length of message and the internet address and local port number of destination socket as follows

Array of bytes containing message	Length of Messages	Internet Address	Local Port Number
---	-----------------------	------------------	----------------------

Instances of datagram packet may be transmitted between processes when one process sends it and another receives it

This class provides another constructor for use when receiving a message. Its arguments specify an array of bytes in which to receive the message and length of the array a received message is put in the DatagramPacket. Packet together with its length and the internet address and port of sending socket. The message can be received from the

DatagramPackets by a means of method getData. The methods getPort and getAddress access the port and internet address.

**DatagramSocket:** This class supports sockets for sending and receiving UDP Datagrams. It provides a constructor that takes a port number as argument for use by the processes that need to use a particular port. It also provides a no argument constructor that allows the system to choose a free local port. These constructors can throw a SocketException

If the port is already in use or a reserved port.

The class DatagramSocket provides methods that include the following

**Send and receive :** These methods are used for transmitting datagrams between a pair of sockets the argument of send is an instance of DatagramPacket containing message and its destination. The argument of receive is an empty .

**setSoTimeout:** this method allows a timeout to be set with a timeout set , the receive message will block for the time specified and then throw an `InterruptedException`.

**Connect:** this method is used for connecting it to a particular remote port and internet address , in which case the socket is only able to send messages to and receive messages from that address.

## Class **DatagramPacket**

[java.lang.Object](#)

└ **java.net.DatagramPacket**

public final class **DatagramPacket**

extends [Object](#)

This class represents a datagram packet.

Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

**Since:**

JDK1.0

### Constructor Summary

[DatagramPacket](#)(byte[] buf, int length)

Constructs a DatagramPacket for receiving packets of length length.

[DatagramPacket](#)(byte[] buf, int length, [InetAddress](#) address, int port)

Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

[DatagramPacket](#)(byte[] buf, int offset, int length)

Constructs a DatagramPacket for receiving packets of length length, specifying an offset into the buffer.

[DatagramPacket](#)(byte[] buf, int offset, int length, [InetAddress](#) address, int port)

Constructs a datagram packet for sending packets of length length with

offset ioffsetto the specified port number on the specified host.

**[DatagramPacket](#)**(byte[] buf, int offset, int length, [SocketAddress](#) address)

Constructs a datagram packet for sending packets of length length with offset ioffsetto the specified port number on the specified host.

**[DatagramPacket](#)**(byte[] buf, int length, [SocketAddress](#) address)

Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

## Method Summary

<a href="#">InetAddress</a>	<a href="#">getAddress()</a> Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
byte[]	<a href="#">getData()</a> Returns the data buffer.
int	<a href="#">getLength()</a> Returns the length of the data to be sent or the length of the data received.
int	<a href="#">getOffset()</a> Returns the offset of the data to be sent or the offset of the data received.
int	<a href="#">getPort()</a> Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
<a href="#">SocketAddress</a>	<a href="#">getSocketAddress()</a> Gets the SocketAddress (usually IP address + port number) of the remote host that this packet is being sent to or is coming from.
void	<a href="#">setAddress(<a href="#">InetAddress</a> iaddr)</a> Sets the IP address of the machine to which this datagram is being sent.
void	<a href="#">setData(byte[] buf)</a> Set the data buffer for this packet.
void	<a href="#">setData(byte[] buf, int offset, int length)</a> Set the data buffer for this packet.

void	<a href="#"><u>setLength</u></a> (int length) Set the length for this packet.
void	<a href="#"><u>setPort</u></a> (int iport) Sets the port number on the remote host to which this datagram is being sent.
void	<a href="#"><u>setSocketAddress</u></a> ( <a href="#"><u>SocketAddress</u></a> address) Sets the SocketAddress (usually IP address + port number) of the remote host to which this datagram is being sent.

### Methods inherited from class java.lang.[Object](#)

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

### Constructor Detail

#### DatagramPacket

```
public DatagramPacket(byte[] buf,
                      int offset,
                      int length)
```

Constructs a DatagramPacket for receiving packets of length length, specifying an offset into the buffer.

The length argument must be less than or equal to buf.length.

#### Parameters:

buf - buffer for holding the incoming datagram.

offset - the offset for the buffer

length - the number of bytes to read.

#### Since:

JDK1.2



## DatagramPacket

```
public DatagramPacket(byte[] buf,  
                      int length)
```

Constructs a DatagramPacket for receiving packets of length length.  
The length argument must be less than or equal to buf.length.

### Parameters:

buf - buffer for holding the incoming datagram.  
length - the number of bytes to read.

---

## DatagramPacket

```
public DatagramPacket(byte[] buf,  
                      int offset,  
                      int length,  
InetAddress address,  
                      int port)
```

Constructs a datagram packet for sending packets of length length with  
offset ioffsetto the specified port number on the specified host.  
The lengthargument must be less than or equal to buf.length.

### Parameters:

buf - the packet data.  
offset - the packet data offset.  
length - the packet data length.  
address - the destination address.  
port - the destination port number.

### Since:

JDK1.2

### See Also:

[InetAddress](#)

---

## DatagramPacket

```
public DatagramPacket(byte[] buf,
                      int offset,
                      int length,
                      SocketAddress address)
    throws SocketException
```

Constructs a datagram packet for sending packets of length length with offset ioffsetto the specified port number on the specified host. The lengthargument must be less than or equal to buf.length.

### Parameters:

buf - the packet data.  
 offset - the packet data offset.  
 length - the packet data length.  
 address - the destination socket address.

### Throws:

[IllegalArgumentException](#) - if address type is not supported  
[SocketException](#)

### Since:

1.4

### See Also:

[InetAddress](#)

---

## DatagramPacket

```
public DatagramPacket(byte[] buf,
                      int length,
                      InetAddress address,
                      int port)
```

Constructs a datagram packet for sending packets of length length to the specified port number on the specified host. The length argument must be less than or equal to buf.length.

**Parameters:**

buf - the packet data.

length - the packet length.

address - the destination address.

port - the destination port number.

---

**DatagramPacket**

```
public DatagramPacket(byte[] buf,  
                      int length,  
                      SocketAddress address)  
    throws SocketException
```

Constructs a datagram packet for sending packets of length length to the specified port number on the specified host. The length argument must be less than or equal to buf.length.

**Parameters:**

buf - the packet data.

length - the packet length.

address - the destination address.

**Throws:**

[IllegalArgumentException](#) - if address type is not supported  
[SocketException](#)

**Since:**

1.4

**See Also:**

[InetAddress](#)

## Method Detail

### getAddress

public [InetAddress](#) **getAddress()**

Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.

**Returns:**

the IP address of the machine to which this datagram is being sent or from which the datagram was received.

**See Also:**

[InetAddress](#), [setAddress\(java.net.InetAddress\)](#)

---

### getPort

public int **getPort()**

Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.

**Returns:**

the port number on the remote host to which this datagram is being sent or from which the datagram was received.

**See Also:**

[setPort\(int\)](#)

---

### getData

public byte[] **getData()**

Returns the data buffer. The data received or the data to be sent starts from the offset in the buffer, and runs for length long.

**Returns:**

the buffer used to receive or send data

**See Also:**

[setData\(byte\[\], int, int\)](#)

---

**getOffset**

public int **getOffset()**

Returns the offset of the data to be sent or the offset of the data received.

**Returns:**

the offset of the data to be sent or the offset of the data received.

**Since:**

JDK1.2

---

**getLength**

public int **getLength()**

Returns the length of the data to be sent or the length of the data received.

**Returns:**

the length of the data to be sent or the length of the data received.

**See Also:**

[setLength\(int\)](#)

---

**setData**

public void **setData**(byte[] buf,  
                  int offset,  
                  int length)

Set the data buffer for this packet. This sets the data, length and offset of the packet.

**Parameters:**

buf - the buffer to set for this packet

offset - the offset into the data

length - the length of the data and/or the length of the buffer used to receive data

**Throws:**

[NullPointerException](#) - if the argument is null

**Since:**

JDK1.2

**See Also:**

[getData\(\)](#), [getOffset\(\)](#), [getLength\(\)](#)

---

**setAddress**

public void setAddress([InetAddress](#) iaddr)

Sets the IP address of the machine to which this datagram is being sent.

**Parameters:**

iaddr - the InetAddress

**Since:**

JDK1.1

**See Also:**

[getAddress\(\)](#)

---

**setPort**

public void **setPort**(int iport)

Sets the port number on the remote host to which this datagram is being sent.

**Parameters:**

ipport - the port number

**Since:**

JDK1.1

**See Also:**

[getPort\(\)](#)

---

**setSocketAddress**

public void **setSocketAddress**([SocketAddress](#) address)

Sets the SocketAddress (usually IP address + port number) of the remote host to which this datagram is being sent.

**Parameters:**

address - the SocketAddress

**Throws:**

[IllegalArgumentExpection](#) - if address is null or is a SocketAddress subclass not supported by this socket

**Since:**

1.4

**See Also:**

[getSocketAddress\(\)](#)

---

**getSocketAddress**

public [SocketAddress](#) **getSocketAddress**()

Gets the SocketAddress (usually IP address + port number) of the remote host that this packet is being sent to or is coming from.

**Returns:**

the SocketAddress

**Since:**

1.4

**See Also:**[setSocketAddress\(java.net.SocketAddress\)](#)

---

**setData**`public void setData(byte[] buf)`

Set the data buffer for this packet. With the offset of this DatagramPacket set to 0, and the length set to the length of buf.

**Parameters:**

buf - the buffer to set for this packet.

**Throws:**

[NullPointerException](#) - if the argument is null.

**Since:**

JDK1.1

**See Also:**[getLength\(\)](#), [getData\(\)](#)

---

**setLength**`public void setLength(int length)`

Set the length for this packet. The length of the packet is the number of bytes from the packet's data buffer that will be sent, or the number of bytes of the packet's data buffer that will be used for receiving data. The length must be lesser or equal to the offset plus the length of the packet's buffer.

**Parameters:**

length - the length to set for this packet.

**Throws:**

[IllegalArgumentException](#) - if the length is negative or if the length is greater than the packet's data buffer length.



**Since:**

JDK1.1

**See Also:**getLength(), setData(byte[], int, int)

java.net

Class DatagramSocket

[java.lang.Object](#)└ **java.net.DatagramSocket****Direct Known Subclasses:**[MulticastSocket](#)

public class DatagramSocket

extends [Object](#)

This class represents a socket for sending and receiving datagram packets. A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

UDP broadcasts sends are always enabled on a DatagramSocket. In order to receive broadcast packets a DatagramSocket should be bound to the wildcard address. In some implementations, broadcast packets may also be received when a DatagramSocket is bound to a more specific address.

Example: DatagramSocket s = new DatagramSocket(null); s.bind(new InetSocketAddress(8888)); Which is equivalent to: DatagramSocket s = new DatagramSocket(8888); Both cases will create a DatagramSocket able to receive broadcasts on UDP port 8888.

**Since:**

JDK1.0

**See Also:**DatagramPacket, DatagramChannel

### Constructor Summary

	<a href="#"><b>DatagramSocket</b></a> () Constructs a datagram socket and binds it to any available port on the local host machine.
protected	<a href="#"><b>DatagramSocket</b></a> ( <a href="#"><b>DatagramSocketImpl</b></a> impl) Creates an unbound datagram socket with the specified <a href="#"><b>DatagramSocketImpl</b></a> .
	<a href="#"><b>DatagramSocket</b></a> (int port) Constructs a datagram socket and binds it to the specified port on the local host machine.
	<a href="#"><b>DatagramSocket</b></a> (int port, <a href="#"><b>InetAddress</b></a> laddr) Creates a datagram socket, bound to the specified local address.
	<a href="#"><b>DatagramSocket</b></a> ( <a href="#"><b>SocketAddress</b></a> bindaddr) Creates a datagram socket, bound to the specified local socket address.

### Method Summary

void	<a href="#"><b>bind</b></a> ( <a href="#"><b>SocketAddress</b></a> addr) Binds this <a href="#"><b>DatagramSocket</b></a> to a specific address & port.
void	<a href="#"><b>close</b></a> () Closes this datagram socket.
void	<a href="#"><b>connect</b></a> ( <a href="#"><b>InetAddress</b></a> address, int port) Connects the socket to a remote address for this socket.
void	<a href="#"><b>connect</b></a> ( <a href="#"><b>SocketAddress</b></a> addr) Connects this socket to a remote socket address (IP address + port number).
void	<a href="#"><b>disconnect</b></a> () Disconnects the socket.
boolean	<a href="#"><b>getBroadcast</b></a> () Tests if <code>SO_BROADCAST</code> is enabled.

<a href="#"><u>DatagramChannel</u></a>	<a href="#"><u>getChannel()</u></a> Returns the unique <a href="#"><u>DatagramChannel</u></a> object associated with this datagram socket, if any.
<a href="#"><u>InetAddress</u></a>	<a href="#"><u>getInetAddress()</u></a> Returns the address to which this socket is connected.
<a href="#"><u>InetAddress</u></a>	<a href="#"><u>getLocalAddress()</u></a> Gets the local address to which the socket is bound.
int	<a href="#"><u>getLocalPort()</u></a> Returns the port number on the local host to which this socket is bound.
<a href="#"><u>SocketAddress</u></a>	<a href="#"><u>getLocalSocketAddress()</u></a> Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
int	<a href="#"><u>getPort()</u></a> Returns the port for this socket.
int	<a href="#"><u>getReceiveBufferSize()</u></a> Get value of the SO_RCVBUF option for this DatagramSocket, that is the buffer size used by the platform for input on thisDatagramSocket.
<a href="#"><u>SocketAddress</u></a>	<a href="#"><u>getRemoteSocketAddress()</u></a> Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
boolean	<a href="#"><u>getReuseAddress()</u></a> Tests if SO_REUSEADDR is enabled.
int	<a href="#"><u>getSendBufferSize()</u></a> Get value of the SO_SNDBUF option for this DatagramSocket, that is the buffer size used by the platform for output on thisDatagramSocket.
int	<a href="#"><u>getSoTimeout()</u></a> Retrive setting for SO_TIMEOUT.
int	<a href="#"><u>getTrafficClass()</u></a> Gets traffic class or type-of-service in the IP datagram header for packets sent from this DatagramSocket.
boolean	<a href="#"><u>isBound()</u></a> Returns the binding state of the socket.

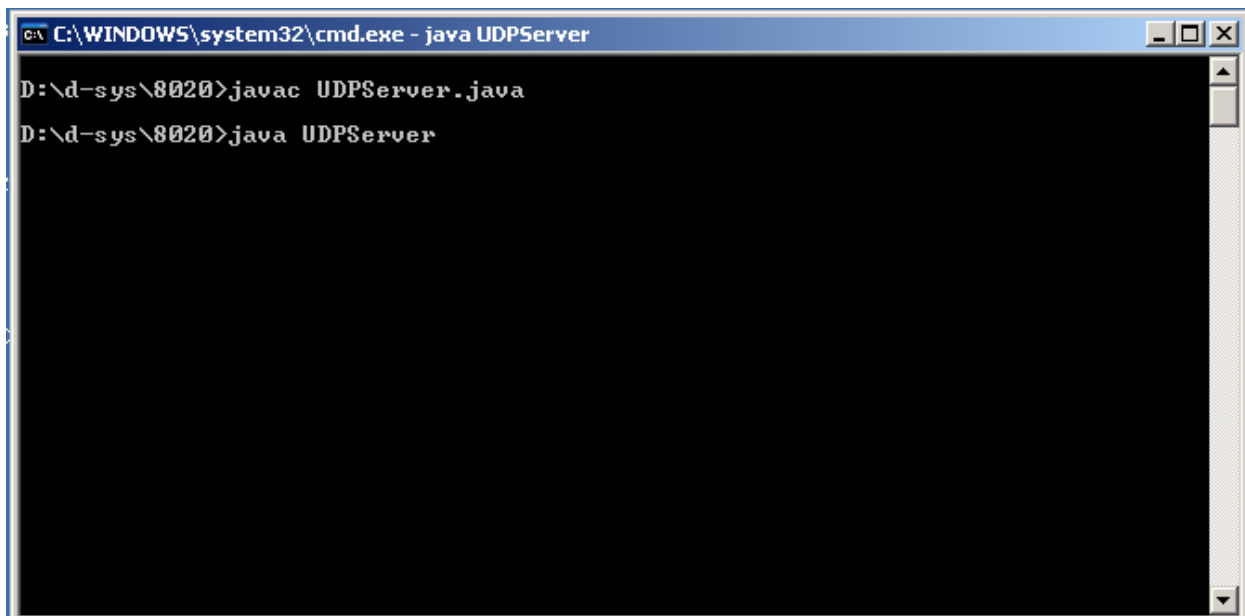
boolean	<a href="#"><b>isClosed()</b></a> Returns whether the socket is closed or not.
boolean	<a href="#"><b>isConnected()</b></a> Returns the connection state of the socket.
void	<a href="#"><b>receive(DatagramPacket p)</b></a> Receives a datagram packet from this socket.
void	<a href="#"><b>send(DatagramPacket p)</b></a> Sends a datagram packet from this socket.
void	<a href="#"><b>setBroadcast(boolean on)</b></a> Enable/disable SO_BROADCAST.
static void	<a href="#"><b>setDatagramSocketImplFactory(DatagramSocketImplFactory fac)</b></a> Sets the datagram socket implementation factory for the application.
void	<a href="#"><b>setReceiveBufferSize(int size)</b></a> Sets the SO_RCVBUF option to the specified value for this DatagramSocket.
void	<a href="#"><b>setReuseAddress(boolean on)</b></a> Enable/disable the SO_REUSEADDR socket option.
void	<a href="#"><b>setSendBufferSize(int size)</b></a> Sets the SO_SNDBUF option to the specified value for this DatagramSocket.
void	<a href="#"><b>setSoTimeout(int timeout)</b></a> Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
void	<a href="#"><b>setTrafficClass(int tc)</b></a> Sets traffic class or type-of-service octet in the IP datagram header for datagrams sent from this DatagramSocket.

**UDPServer:**

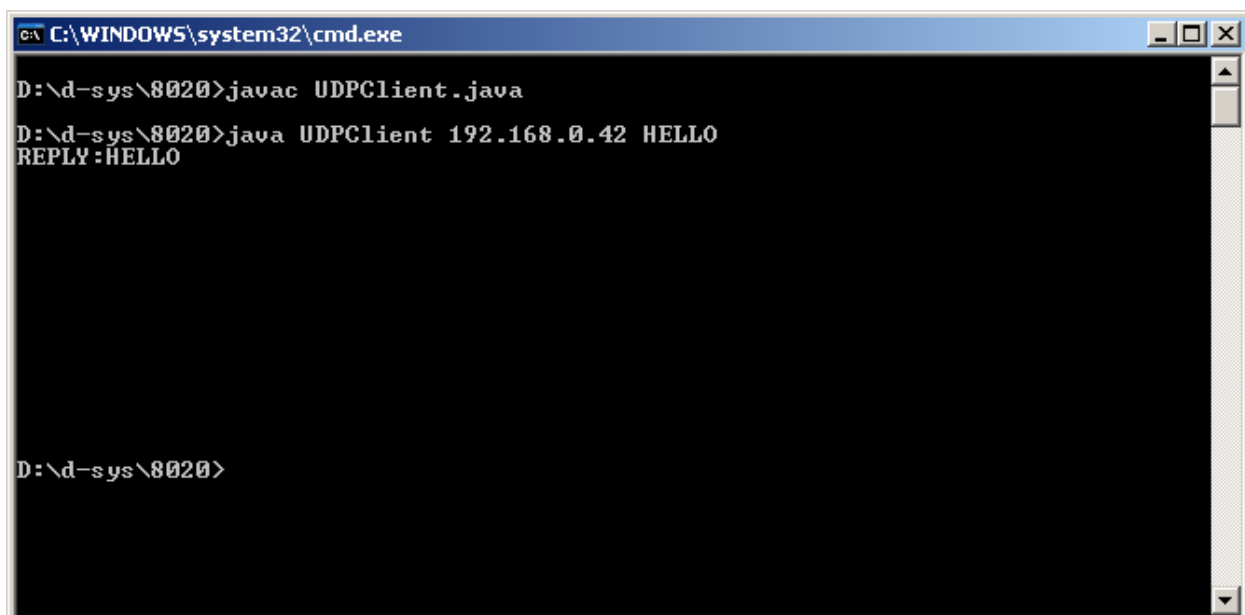
```
import java.net.*;
import java.io.*;
public class UDPServer
{
    public static void main(String args[])
    {
        DatagramSocket server=null;
        try
        {
            server=new DatagramSocket(6789);
            byte[] buffer=new byte[1000];
            while(true)
            {
                DatagramPacket request=new DatagramPacket(buffer,buffer.length);
                server.receive(request);
                DatagramPacket reply=new
                DatagramPacket(request.getData(),request.getLength(),request.getAddress(),request.getPort());
                server.send(reply);
            }
        }
        catch(SocketException e)
        {
            System.out.println("Socket!" + e.getMessage());
        }
        catch(IOException e)
        {
            System.out.println("IO:" + e.getMessage());
        }
        finally
        {
            if(server!=null)
                server.close();
        }
    }
}
```

**UDPClient:**

```
import java.net.*;
import java.io.*;
public class UDPClient
{
    public static void main(String args[])
    {
        DatagramSocket client=null;
        try
        {
            client=new DatagramSocket();
            byte[] m=args[1].getBytes();
            InetAddress host=InetAddress.getByName(args[0]);
            int serverport=6789;
            DatagramPacket request=new DatagramPacket(m,args[1].length(),host,serverport);
            client.send(request);
            byte[] buffer=new byte[1000];
            DatagramPacket reply=new DatagramPacket(buffer,buffer.length);
            client.receive(reply);
            String re=new String(reply.getData());
            System.out.println("REPLY:"+re);
        }
        catch(SocketException e)
        {
            System.out.println("SoCKET!" +e.getMessage());
        }
        catch(IOException e)
        {
            System.out.println("IO:" +e.getMessage());
        }
        finally
        {
            if(client!=null)client.close();
        }
    }
}
```

**OUTPUT:**

```
C:\WINDOWS\system32\cmd.exe - java UDPServer
D:\d-sys\8020>javac UDPServer.java
D:\d-sys\8020>java UDPServer
```



```
C:\WINDOWS\system32\cmd.exe
D:\d-sys\8020>javac UDPCClient.java
D:\d-sys\8020>java UDPCClient 192.168.0.42 HELLO
REPLY:HELLO

D:\d-sys\8020>
```

## **6. IMPLEMENTATION OF BULLETIN BOARD.**

### **BULLETIN BOARD:**

A Bulletin Board System, or BBS, is a computer system running software that allows users to connect and log in to the system using a terminal program. Once logged in, a user can perform functions such as uploading and downloading software and data, reading news and bulletins, and exchanging messages with other users, either through electronic mail or in public message boards. Many BBSes also offer on-line games, in which users can compete with each other, and BBSes with multiple phone lines often provide chat rooms, allowing users to interact with each other.

Although a BBS is typically an acronym for Bulletin Board System, it is sometimes also referred to as a Bulletin Board Service, which would clearly be an acceptable answer if the matter ever came up in Trivial Pursuit.

Originally BBSes were accessed only over a phone line using a modem, but by the early 1990s some BBSes allowed access via a Telnet, packet switched network, or packet radio connection.

Ward Christensen coined the term "Bulletin Board System" as a reference to the traditional cork-and-pin bulletin board often found in entrances of supermarkets, schools, libraries or other public areas where people can post messages, advertisements, or community news. By "computerizing" this method of communications, the name of the system was born: CBBS - Computerized Bulletin Board System. See History.

During their heyday from the late 1970s to the mid 1990s, most BBSes were run as a hobby free of charge by the system operator (or "SysOp"), while other BBSes charged their users a subscription fee for access, or were operated by a business as a means of supporting their customers. Bulletin Board Systems were in many ways a precursor to the modern form of the World Wide Web and other aspects of the Internet.

Early BBSes were often a local phenomenon, as one had to dial into a BBS with a phone line and would have to pay additional long distance charges for a BBS out of the local calling area. Thus, many users of a given BBS usually lived in the



same area, and activities such as BBS Meets or Get Togethers, where everyone from the board would gather and meet face to face, were common.

As the use of the Internet became more widespread in the mid to late 1990s, traditional BBSes rapidly faded in popularity. Today, Internet forums occupy much of the same social and technological space as BBSes did, and the term BBS is often used to refer to any online forum or message board.

Although BBSing survives only as a niche hobby in most parts of the world, it is still an extremely popular form of communication for Taiwanese youth (see PTT Bulletin Board System). Most BBSes are now accessible over telnet and typically offer free email accounts, FTP services, IRC and all of the protocols commonly used on the Internet.

Most early BBSes operated as stand-alone islands. Information contained on that BBS never left the system, and users would only interact with the information and user community on that BBS alone. However, as BBSes became more widespread, there evolved a desire to connect systems together to share messages and files with distant systems and users. The largest such network was FidoNet.

As it was prohibitively expensive for the hobbyist SysOp to have a dedicated connection to another system, FidoNet was developed as a store and forward network. Private electronic mail (Netmail), public message boards (Echomail) and eventually even file attachments on a FidoNet-capable BBS would be bundled into one or more archive files over a set time interval. These archive files were then compressed with ARC or ZIP and forwarded to (or polled by) another nearby node or hub via a dialup Xmodem session. Messages would be relayed around various FidoNet hubs until they were eventually delivered to their destination. The hierarchy of FidoNet BBS nodes, hubs, and zones was maintained in a routing table called a Nodelist. Some larger BBSes or regional FidoNet hubs would make several transfers per day, some even to multiple nodes or hubs, and as such, transfers usually occurred at night or early morning when toll rates were lowest. In Fido's heyday, sending a Netmail message to a user on a distant FidoNet node, or participating in an Echomail discussion could take days, especially if any FidoNet nodes or hubs in the message's route only made one transfer call per day.

**bulletinserver:**

```
import java.util.Properties;
import java.util Enumeration;
import java.io.*;
import java.net.*;
import java.awt.*;
public class bulletinserver
{
    ServerSocket server;
    Socket connection;
    DataOutputStream output;
    DataInputStream input;
    String s;
    Properties clients,bulletin;
    FileInputStream fcin=null,fbbin=null;
    FileOutputStream fcout=null,fbbout=null;
    Enumeration e;
    public bulletinserver()
    {
        try
        {
            clients=new Properties();
            bulletin=new Properties();
            try
            {
                fcin=new FileInputStream("NameList.dat");
                fbbin=new FileInputStream("bulletin.dat");
                if(fcin!=null)
                {
                    clients.load(fcin);
                    fcin.close();
                }
                if(fbbin!=null)
                {
                    bulletin.load(fbbin);
                    fbbin.close();
                }
            }
            catch(IOException ex)
```

```

        {
            ex.printStackTrace();
        }
        server=new ServerSocket(5000,100);
        connection=server.accept();
        System.out.println("Connection established");
        output=new DataOutputStream(connection.getOutputStream());
        input=new DataInputStream(connection.getInputStream());
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
}

public void runServer()
{
    while(true)
    {
        try
        {
            s=input.readUTF();
            if(s.equals("connect"))
            {
                String s1=input.readUTF();
                String s2=input.readUTF();
                boolean bres=connect(s1,s2);
                if(bres==true)
                    output.writeUTF("valid user");
                else
                    output.writeUTF("invalid user");
            }
            if(s.equals("showbulletinboard"))
            {
                String s=showbulletinboard();
                output.writeUTF(s);
            }
            if(s.equals("updatebuffer"))
            {
                String s1=input.readUTF();
                String s2=input.readUTF();

```

```

        boolean b=updatebuffer(s1,s2);
        if(b==true)
            output.writeUTF("message appended");
        else
            output.writeUTF("Not able to write");
        }
        if(s.equals("close"))
        {
            System.exit(0);
        }
        catch(IOException e)
        {
            e.printStackTrace();
        }
    }
    public boolean connect(String user,String pwd)
    {
        String user1=(String)clients.get(user);
        String pwd1=(String)clients.get(pwd);
        if(user1==null&&pwd1==null)
        {
            return false;
        }
        return true;
    }
    public boolean updatebuffer(String name,String msg)
    {
        bulletin.put(name,msg);
        try
        {
            fbbout=new FileOutputStream("bulletin.dat");
            bulletin.store(fbbout,"Bulletin Board");
            fbbout.close();
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
    }

```

```

        return true;
    }
    public String showbulletinboard()
    {
        try
        {
            fbbin=new FileInputStream("bullrtin.dat");
            bulletin.load(fbbin);
            fbbin.close();
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
        Enumeration e=bulletin.keys();
        String[] bulletinList=new String[bulletin.size()];
        int i=0;
        while(e.hasMoreElements())
        {
            bulletinList[i++]=(String) e.nextElement();
        }
        StringBuffer sb=new StringBuffer();
        for(i=0;i<bulletinList.length;i++)
        {
            sb.append("\n"+bulletinList[i]+":"+bulletin.get(bulletinList[i]));
        }
        String sp=sb.toString();
        return sp;
    }
    public boolean addHost(String id,String name)
    {
        if(clients.get(name)!=null)
        {
            return false;
        }
        clients.put(name,id);
        try
        {
            fcout=new FileOutputStream("NameList.dat");
            clients.store(fcout,"NameSpace");
        }
    }

```

```

        fcout.close();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
    return true;
}
public static void main(String args[])
{
    int value;
    String s1,s2;
    bulletinserver bs=new bulletinserver();
    try
    {
        DataInputStream din=new DataInputStream(System.in);
        System.out.println("Do u want to add a new host");
        System.out.println("Type1 to add a host name");
        System.out.println("To run a srver Type2");
        String s=din.readLine();
        value=Integer.parseInt(s);
        while(value==1)
        {
            System.out.println("Enter user name:");
            s1=din.readLine();
            System.out.println("Enter passwd:");
            s2=din.readLine();
            boolean b=bs.addHost(s1,s2);
            if(b)
                System.out.println("Hostname is added");
            else
                System.out.println("Hostname alrady exist");
            System.out.println("Do u want to add a new host");
            System.out.println("Type1 to add a Hostname");
            s=din.readLine();
            value=Integer.parseInt(s);
        }
        System.out.println("server is running");
        bs.runServer();
    }
}

```

```

        catch(Exception e)
        {}
    }
}

```

### **bulletinclient:**

```

import java.util.*;
import java.io.*;
import java.net.*;
import java.awt.*;
public class bulletinclient extends Frame
{
    TextArea display;
    TextField t1,t2;
    Label l1,l2;
    Button send,show,connect,close;
    Panel p1,p2;
    Properties pc;
    Socket client;
    DataInputStream input;
    DataOutputStream output;
    String s;
    public bulletinclient(String s1)
    {
        super("Client");
        pc=new Properties();
        t1=new TextField(20);
        t2=new TextField(20);
        l1=new Label("Enter username");
        l2=new Label("Enter password");
        p2=new Panel();
        p2.add(l1);
        p2.add(t1);
        p2.add(l2);
        p2.add(t2);
        close=new Button("Close");
        show=new Button("bulletinboard");
        send=new Button("send message");
        connect=new Button("Login");
    }
}

```

```

p1=new Panel();
p1.add(connect);
p1.add(show);
p1.add(send);
p1.add(close);
display=new TextArea(20,10);
add("Center",display);
add("South",p1);
add("North",p2);
setSize(600,400);
setVisible(true);
try
{
client=new Socket(s1,5000);
display.append("create socket\n");
display.append("\n Type2 at serverside before running client\n");
input=new DataInputStream(client.getInputStream());
output=new DataOutputStream(client.getOutputStream());
}
catch(IOException ex)
{
ex.printStackTrace();
}
}
public boolean action(Event event,Object o)
{
try
{
if(event.target==connect)
{
output.writeUTF("connect");
output.writeUTF(t1.getText());
output.writeUTF(t2.getText());
s=input.readUTF();
if(s.equals("valid user"))
{
l1.setText("Articlename:");
l2.setText("Articledetails:");
t1.setText(" ");
t2.setText(" ");

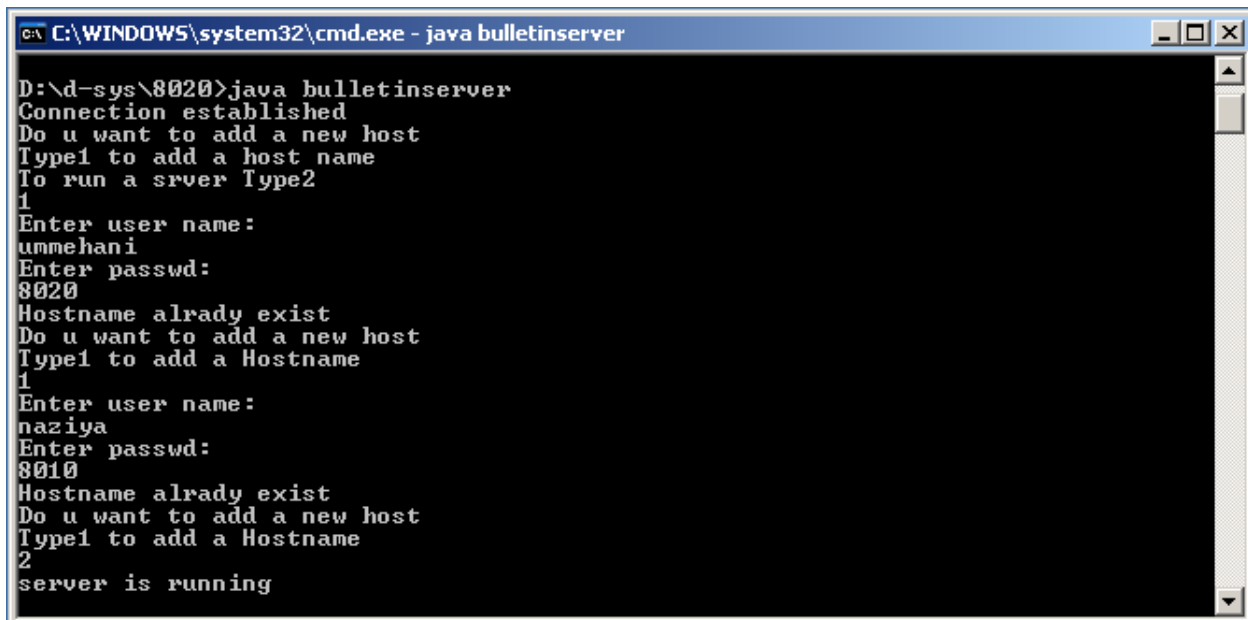
```



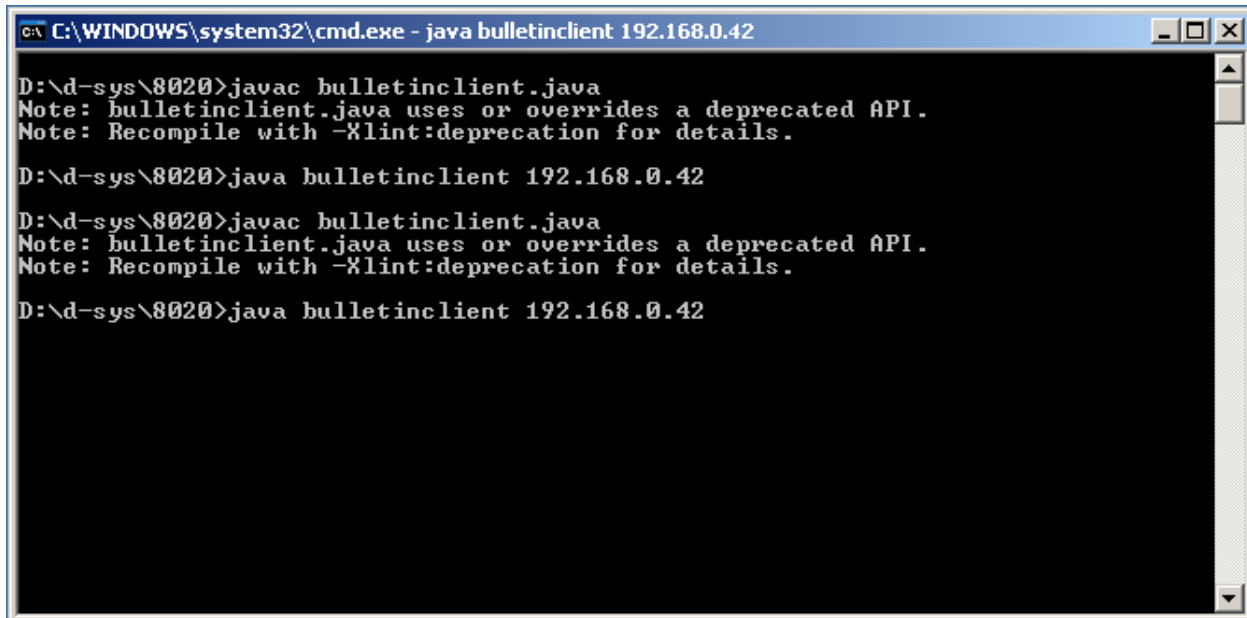
```
display.appendText("\n conformation message:"+s);
}
else
{
display.appendText("\n Invalid user");
}
}
if(event.target==send)
{
if(s.equals("valid user"))
{
output.writeUTF("updatebuffer");
output.writeUTF(t1.getText());
output.writeUTF(t2.getText());
String s1=input.readUTF();
display.setText("\n"+s1);
}
else
{
display.setText("invalid user...cannot post articles");
}
}
if(event.target==show)
{
if(s.equals("valid user"))
{
output.writeUTF("showbulletinboard");
display.setText(" ");
String s=input.readUTF();
display.setText("\n"+s);
}
else
{
display.setText("invalid user...cannot post articles");
}
}
if(event.target==close)
{
output.writeUTF("close");
System.exit(0);
}
```

```
}  
}  
catch(IOException e)  
{  
    e.printStackTrace();  
}  
return true;  
}  
public static void main(String args[])  
{  
    bulletinclient bc=new bulletinclient(args[0]);  
}  
}
```

### OUTPUT:



```
C:\WINDOWS\system32\cmd.exe - java bulletinserver  
D:\d-sys\8020>java bulletinserver  
Connection established  
Do u want to add a new host  
Type1 to add a host name  
To run a srver Type2  
1  
Enter user name:  
ummehani  
Enter passwd:  
8020  
Hostname alrady exist  
Do u want to add a new host  
Type1 to add a Hostname  
1  
Enter user name:  
naziya  
Enter passwd:  
8010  
Hostname alrady exist  
Do u want to add a new host  
Type1 to add a Hostname  
2  
server is running
```



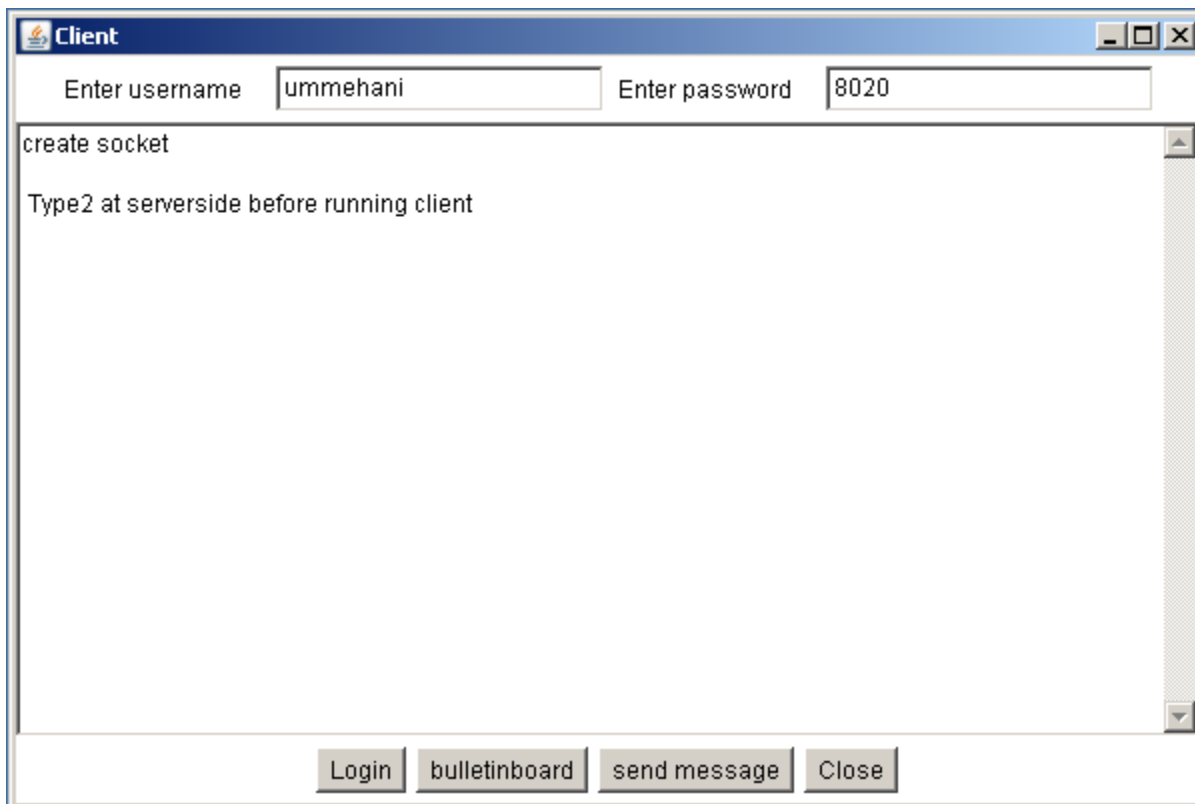
```
C:\WINDOWS\system32\cmd.exe - java bulletinclient 192.168.0.42

D:\d-sys\8020>javac bulletinclient.java
Note: bulletinclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\d-sys\8020>java bulletinclient 192.168.0.42

D:\d-sys\8020>javac bulletinclient.java
Note: bulletinclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\d-sys\8020>java bulletinclient 192.168.0.42
```



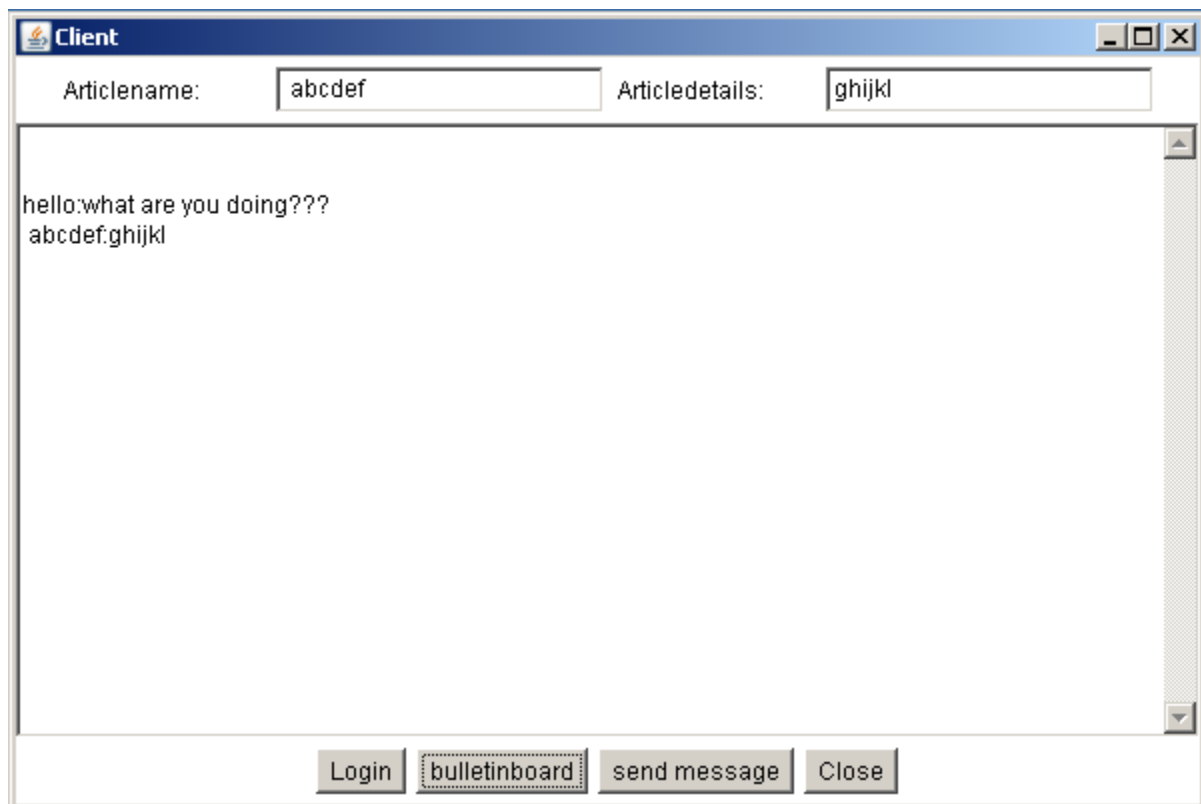
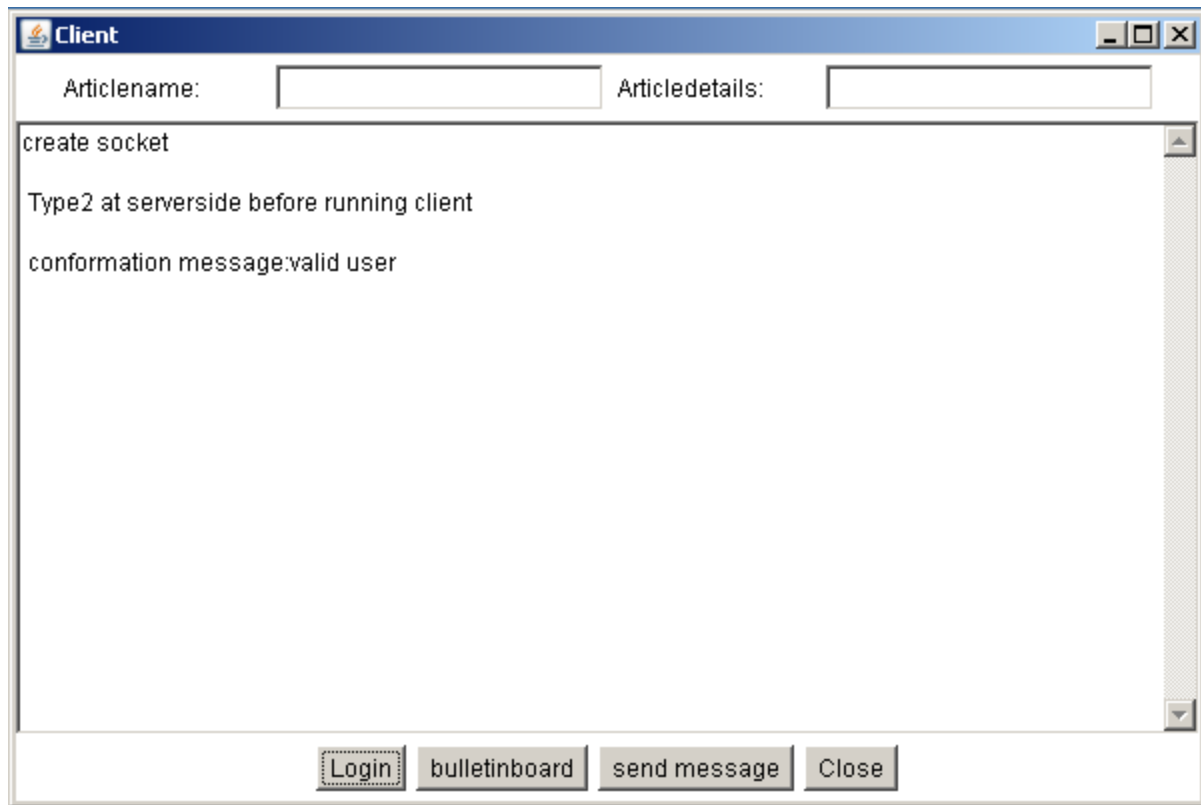
Client

Enter username  Enter password

create socket

Type2 at serverside before running client

Login bulletinboard send message Close



## 7.DEMONSTRATION OF SIMPLE CLIENT SERVER COMMUNICATION

### **simple client server communication:**

In Computer science client-server is a software architecture model consisting of two parts, client systems and server systems, both communicate over a computer network or on the same computer. A client-server application is a distributed system consisting of both client and server software. The client process always initiates a connection to the server, while the server process always waits for requests from any client. When both the client process and server process are running on the same computer, this is called a single seat setup.

Another type of related software architecture is known as peer-to-peer, because each host or application instance can simultaneously act as both a client and a server (unlike centralized servers of the client-server model) and because each has equivalent responsibilities and status. Peer-to-peer architectures are often abbreviated using the acronym P2P.

The client-server relationship describes the relation between the client and how it makes a service request from the server, and how the server can accept these requests, process them, and return the requested information to the client. The interaction between client and server is often described using sequence diagrams. Sequence diagrams are standardized in the Unified Modeling Language.

Both client-server and P2P architectures are in wide usage today.

The basic type of client-server architecture employs only two types of hosts: clients and servers. This type of architecture is sometimes referred to as two-tier. The two-tier architecture means that the client acts as one tier and server process acts as the other tier.

The client-server architecture has become one of the basic models of network computing. Many types of applications have being written using the client-server model. Standard networked functions such as E-mail exchange, web access and database access, are based on the client-server model. For example, a web browser is a client program at the user computer that may access information at any web server in the world.

**Clients characteristics:**

- Always initiates requests to servers.
- Waits for replies.
- Receives replies.
- Usually connects to a small number of servers at one time.
- Usually interacts directly with end-users using any user interface such as graphical user interface.

**Server characteristics:**

- Always wait for a request from one of the clients.
- Serve clients requests then replies with requested data to the clients.
- A server may communicate with other servers in order to serve a client request.

**Advantages:**

- In most cases, a client-server architecture enables the roles and responsibilities of a computing system to be distributed among several independent computers that are known to each other only through a network, so one of advantages of this model is greater ease of maintenance. For example, it is possible to replace, repair, upgrade, or even relocate a server while its clients remain both unaware and unaffected by that change. This independence from change is also referred to as encapsulation.
- All the data is stored on the servers, which generally have much security controls than most clients. Servers can better control access and resources, to guarantee that only those clients with the appropriate permissions may access and change data.
- Since data storage is centralized, updates to that data are much easier to administrators than what would be possible under a P2P architecture. Under a P2P architecture, data updates may need to be distributed and applied to each "peer" in the network, which is both time-consuming and error-prone, as there can be thousands or even millions of peers.
- Many advanced client-server technologies are already available which were designed to ensure security, user friendly interfaces, and ease of use.
- It works with multiple different clients of different specifications.

**Disadvantages:**

- Networks traffic blocking is one of the problems related to the client-server model. As the number of simultaneous client requests to a given server increases, the server can become overloaded. Contrast that to a P2P network, where its bandwidth actually increases as more nodes are added, since the P2P network's overall bandwidth can be roughly computed as the sum of the bandwidths of every node in that network.
- Comparing client-server model to the P2P model, if one server fail, clients' requests cannot be served but in case of P2P networks, servers are usually distributed among many nodes. Even if one or more nodes fail, for example if a node failed to download a file the remaining nodes should still have the data needed to complete the download.

**Server:**

```
import java.io.*;
import java.net.*;
import java.awt.*;
public class Server extends Frame{
    TextArea display;
    Label l;
    TextField t1;
    Panel p1,p2;
    Button b1,close;
    ServerSocket server;
    Socket connection;
    DataOutputStream output;
    DataInputStream input;
    String s;
    public Server(){
        super("server");
        display=new TextArea(20,5);
        l=new Label("enter text to send");
        t1=new TextField(20);
        p1=new Panel();
        p2=new Panel();
        b1=new Button("send");
        close=new Button("close");
        p2.add(b1);
        p2.add(close);
        p1.add(l);
        p1.add(t1);
        add("Center", display);
        add("North",p1);
        add("South",p2);
        resize(300,150);
        show();
    }
    try{
        server=new ServerSocket(5000,100);
        connection=server.accept();
        display.setText("connection rec...\n");
        output=new DataOutputStream(connection.getOutputStream());
        input=new DataInputStream(connection.getInputStream());
```



```

    }
    catch(IOException e){e.printStackTrace();}
    }
    public boolean action(Event e,Object o)
    {
    try{
    if(e.target==b1)
    {
    s=t1.getText();
    display.appendText("\n data sent:\n"+s);
    output.writeUTF(s);
    }
    if(e.target==close)
    {
    display.appendText("Transmission complete.closing socket \n");
    connection.close();
    System.exit(0);
    }
    }catch(IOException ex){ex.printStackTrace();}
    return true;
    }
    public void runServer()
    {
    while(true){
    try{
        s=input.readUTF();
        display.appendText("\n from client \n" +s);
    }catch(IOException e){e.printStackTrace();}
    }
    }
    public static void main (String args[ ]){
    Server s=new Server();
    s.runServer();
    }
    }

```

**Client:**

```

import java.io.*;
import java.net.*;
import java.awt.*;
public class Client extends Frame{
    TextArea display;
    TextField t1;
    Label l1;
    Button close,send;
    Panel p1,p2;
    String s;
    Socket client;
    DataOutputStream output;
    DataInputStream input;
    public Client(){
        super("client");
        t1=new TextField(20);
        l1=new Label("Enter Text");
        p2=new Panel();
        p2.add(l1);
        p2.add(t1);
        close=new Button("close");
        send=new Button("send");
        p1=new Panel();
        p1.add(send);
        p2.add(close);
        display=new TextArea(20,10);
        add("Center", display);
        add("South",p1);
        add("North",p2);
        resize(300,150);
        show();
        try{
            client =new Socket(InetAddress.getLocalHost(),5000);
            display.appendText("created socket \n");
            input=new DataInputStream(client.getInputStream());
            output=new DataOutputStream(client.getOutputStream());
        }
        catch(IOException ex){ex.printStackTrace();}
    }
}

```

```
}
public boolean action(Event event, Object o){
try{
if(event.target==close){
client.close();
System.exit(0);
}
if(event.target==send)
{
s=t1.getText();
output.writeUTF(s);
display.appendText("\n message sent:"+s);
}
}
catch(IOException e){e.printStackTrace();}
return true;
}
public void runClient(){
while(true){
try{
s=input.readUTF();
display.appendText("\n from Server:"+s);
}
catch(IOException e){e.printStackTrace();}
}
}
public static void main(String args[])
{
Client c=new Client();
c.runClient();
}
}
```

**OUTPUT:**

```

C:\WINDOWS\system32\cmd.exe - java Server
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dcet>d:

D:\>cd d-sys

D:\d-sys>cd 8020

D:\d-sys\8020>javac server.java
Note: server.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\d-sys\8020>javac Server.java
Note: Server.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\d-sys\8020>java Server
_

```

```

C:\WINDOWS\system32\cmd.exe - java Client
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dcet>d:

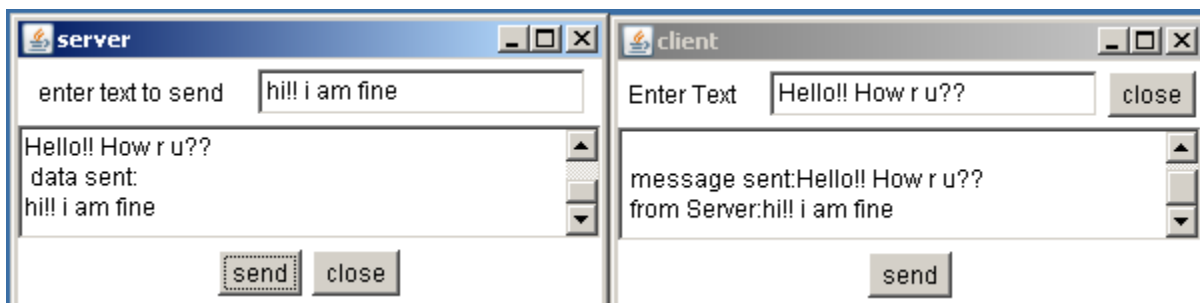
D:\>cd d-sys

D:\d-sys>cd 8020

D:\d-sys\8020>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\d-sys\8020>java Client

```



## 8. CHATTING APPLICATION USING TCP

### Server:

```
import java.io.*;
import java.net.*;
import java.lang.*;

class TcpMyServer1
{
    public static void main(String[] args)
    {
        try
        {
            String str1=" ";
            String str2=" ";
            ServerSocket ss=new ServerSocket(5678);
            System.out.println("Server Socket Created"+ss);
            Socket s=ss.accept();
            System.out.println("Connection Created"+s);
            BufferedReader br1,br2;
            PrintWriter pw;
            br1=new BufferedReader(new InputStreamReader(System.in));
            br2=new BufferedReader(new
InputStreamReader(s.getInputStream()));
            pw= new PrintWriter(s.getOutputStream(),true);
            while(true)
            {

                System.out.println("Enter the data to send");
                str1=br1.readLine();
                pw.println(str1);
                str2=br2.readLine();
                System.out.println("The Received Data "+str2);
                if(str2.equals("bye"))
                    break;
            }
        }
        catch(SocketException e)
        {
```

```

        System.out.println(e);
    }
    catch(IOException e)
    {
        System.out.println(e);
    }
}
}

```

### **Client:**

```

import java.io.*;
import java.net.*;
import java.lang.*;

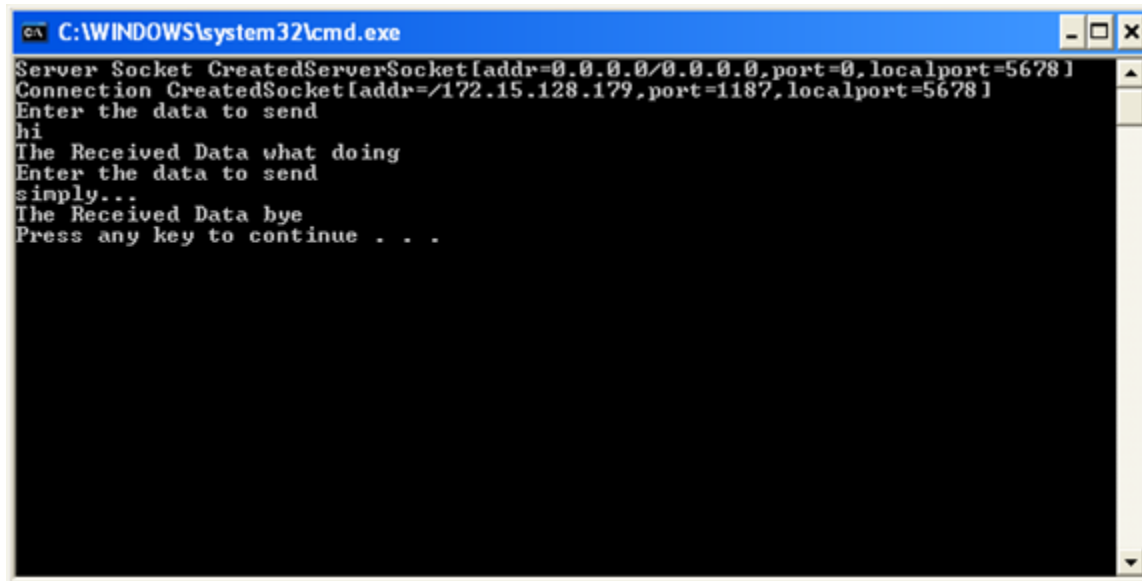
class TcpMyClient1
{
    public static void main(String[] args)
    {
        String str1=" ";
        String str2=" ";
        BufferedReader br1,br2;
        PrintWriter pw;
        Socket s;
        try
        {
            InetAddress ia=InetAddress.getLocalHost();
            s=new Socket(ia,5678);
            System.out.println("Socket Created"+s);
            br1=new BufferedReader(new InputStreamReader(System.in));
            br2=new BufferedReader(new
InputStreamReader(s.getInputStream()));
            pw= new PrintWriter(s.getOutputStream(),true);
            while(true)
            {
                str1=br2.readLine();
                System.out.println("The Received Data :"+ str1);
                if(str1.equals("bye"))

```

```
        break;
        System.out.println("Enter the data to send");
        str2=br1.readLine();
        pw.println(str2);
    }
}
catch(SocketException e)
{
    System.out.println(e);
}
catch(IOException e)
{
    System.out.println(e);
}
}
```

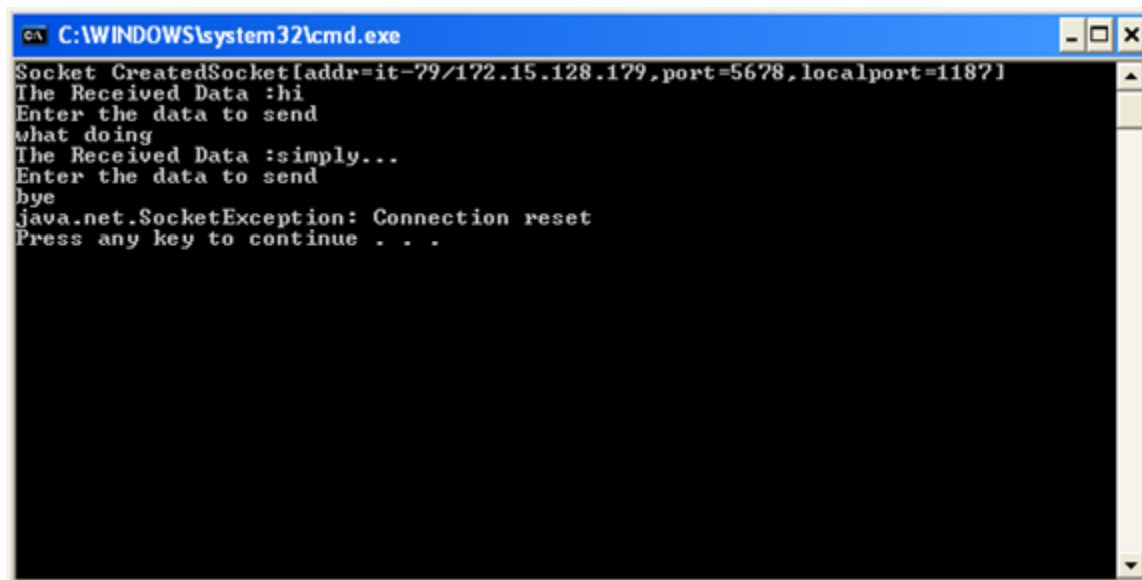
**OUTPUT:**

**SERVER:**



```
C:\WINDOWS\system32\cmd.exe
Server Socket CreatedServerSocket[addr=0.0.0.0/port=0,localport=5678]
Connection CreatedSocket[addr=/172.15.128.179,port=1187,localport=5678]
Enter the data to send
hi
The Received Data what doing
Enter the data to send
simply...
The Received Data bye
Press any key to continue . . .
```

**CLIENT:**



```
C:\WINDOWS\system32\cmd.exe
Socket CreatedSocket[addr=it-79/172.15.128.179,port=5678,localport=1187]
The Received Data :hi
Enter the data to send
what doing
The Received Data :simply...
Enter the data to send
bye
java.net.SocketException: Connection reset
Press any key to continue . . .
```



## 9. CHATTING APPLICATION USING UDP

### Server:

```
import java.io.*;
import java.net.*;
import java.lang.*;

class MyServer
{
    public static void main(String[] args)
    {
        try{
            String str1,str2;
            DatagramSocket ds;
            DatagramPacket dp1,dp2;
            InetAddress ia;
            byte[] data1=new byte[1024]; byte[] data2=new byte[1024];
            BufferedReader br1;
            ds=new DatagramSocket(1500);
            ia=InetAddress.getLocalHost();
            br1=new BufferedReader(new InputStreamReader(System.in));
            while(true)
            {

                System.out.println("Enter the data to send");
                str1=br1.readLine();
                data1=str1.getBytes();
                dp1=new DatagramPacket(data1,data1.length,ia,1700);
                ds.send(dp1);
                if(str1.equals("bye"))
                    break;
                dp2=new DatagramPacket(data2,data2.length);
                ds.receive(dp2);
                str2=new String(dp2.getData(),0,dp2.getLength());
                System.out.println("The Received Data "+str2);
                if(str2.equals("bye"))
                    break;
            }
        }
    }
}
```

```

    }
    catch(BindException e)
    {
        System.out.println(e);
    }
    catch(SocketException e)
    {
        System.out.println(e);
    }
    catch(IOException e)
    {
        System.out.println(e);
    }
}
}

```

### **Client:**

```

import java.io.*;
import java.net.*;

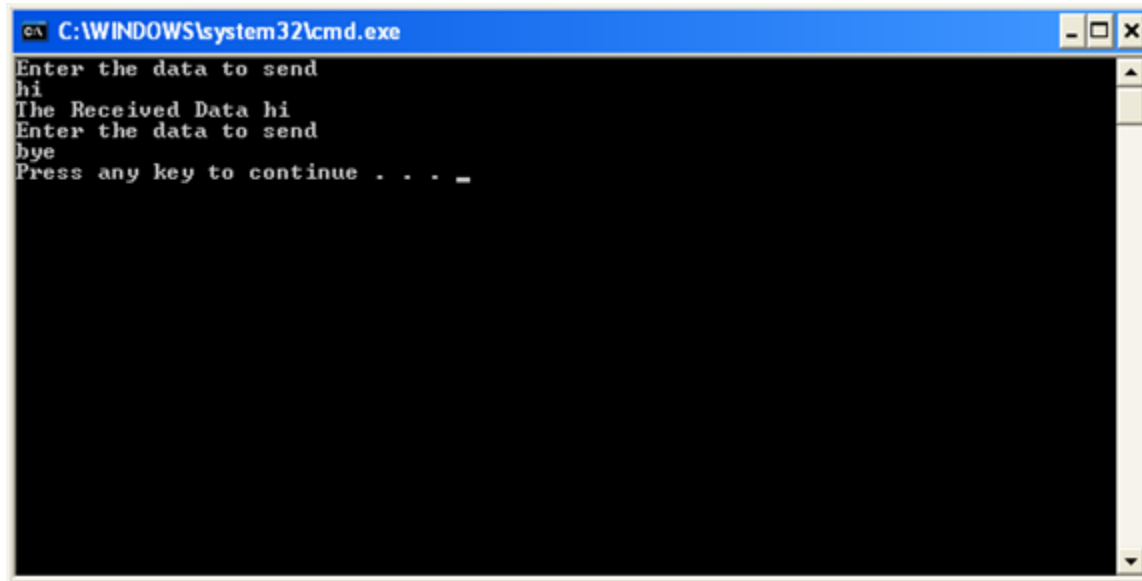
class MyClient
{
    public static void main(String[] args)
    {
        String str1,str2;
        DatagramSocket ds;
        DatagramPacket dp1,dp2;
        InetAddress ia;
        byte[] data1=new byte[1024]; byte[] data2=new byte[1024];
        BufferedReader br;
        try
        {
            ds=new DatagramSocket(1700);
            ia=InetAddress.getLocalHost();
            br=new BufferedReader(new InputStreamReader(System.in));

```

```
        while(true)
        {
            dp1=new DatagramPacket(data1,data1.length);
            ds.receive(dp1);
            str1=new String(dp1.getData(),0,dp1.getLength());
            System.out.println("Length is "+ dp1.getLength());
            System.out.println("The Received Data "+ str1);
            if(str1.equals("bye"))
                break;
            System.out.println("Enter the data to send");
            str2=br.readLine();
            data2=str2.getBytes();
            dp1=new DatagramPacket(data2,data2.length,ia,1500);
            ds.send(dp1);
        }
    }
    catch(BindException e)
    {
        System.out.println(e);
    }
    catch(SocketException e)
    {
        System.out.println(e);
    }
    catch(IOException e)
    {
        System.out.println(e);
    }
}
}
```

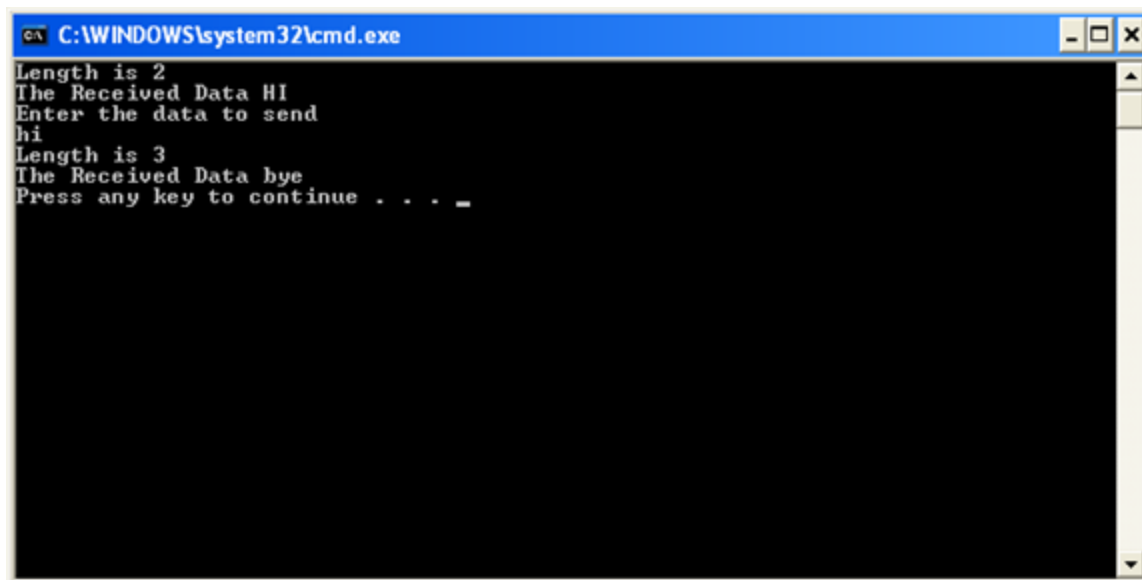
**OUTPUT:**

**SERVER:**



```
C:\WINDOWS\system32\cmd.exe
Enter the data to send
hi
The Received Data hi
Enter the data to send
bye
Press any key to continue . . . _
```

**CLIENT:**



```
C:\WINDOWS\system32\cmd.exe
Length is 2
The Received Data HI
Enter the data to send
hi
Length is 3
The Received Data bye
Press any key to continue . . . _
```