



# Matrusri Engineering College

(Approved by AICTE, Affiliated to Osmania University)

# 16-1-486, Saidabad, Hyderabad-500059. Ph: 040-24072764

## Department of Computer Science & Engineering

email: hodcse@matrusri.edu.in

Website: www.matrusri.edu.in

### B.E. IV/IV CSE SEM-I DISTRIBUTED SYSTEMS LAB

SNo	Name of the program	CO
1	FTP implementation	
2	Iterative Name Server Implementation Using Java	
3	Iterative Name Server Implementation Using Java Server Pages	
4	Chat Server Implementation	
5	Chat Server Implementation Using JSP	
6	Understanding Working of NFS	
7	Bulletin Board – Implementation	
8	Bulletin Board Implementation Using JSP	
9	Implement client-server communication using UDP	
10	Implement client-server communication using TCP	
ADDITIONAL PROGRAMS		
11	RMI program to implement arithmetic operations with GUI	
12	RMI program to implement Domain Name System (DNS)	

# **B.E. IV/IV CSE SEM-I**

## **DISTRIBUTED SYSTEMS LAB**

### **Introduction**

Distributed Systems lab Sessions consists of programs related to Designing, Programming and creating services related to application distributed systems. In this lab we are going to develop applications related to distributed system which includes implementation of FTP protocol, establishing Client-server communication, creating DNS and creating Bulletin-board application.

This lab session consists of programs to develop FTP application which is used to implement FTP protocol manually. In this we are developing simple chat server-chat client application which enables users to chat from different systems. In the lab sessions we are developing simple DNS application with in LAN. And finally we are developing Simple Bulletin-board application which enables group communication.

### **Hardware and software Requirements**

#### **Hardware:**

Processor : Intel P-IV processor

Memory : 1 GB

Hard-disk : 40 GB

Other : Keyboard, Mouse, Monitor

#### **Software:**

Editors : Editplus, Notepad

Softwares : Java Development kit 1.5, Internet Explorer 1.6, FTP

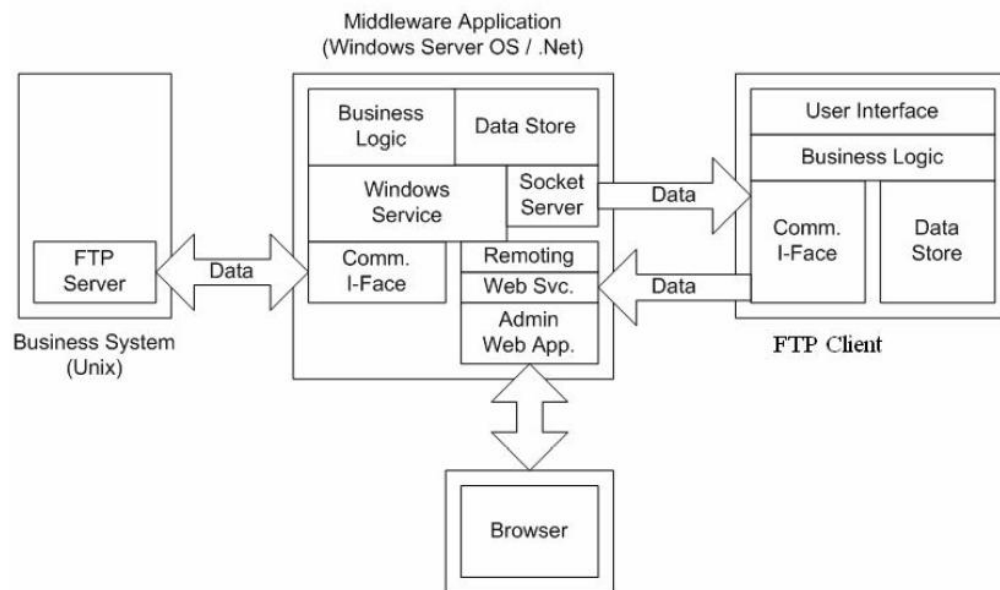
Operating Systems: Windows XP, UNIX, Windows NT

Databases : Ms-Access, ORACLE 8.0

# FTP Client Implementation

## Introduction:

The FTP (File Transfer Protocol) is commonly used for copying files to and from other computers. These computers may be at the same site or at different sites thousands of miles apart. FTP is a general protocol that works on UNIX systems as well as a variety of other (non-UNIX) systems. A user interface for the standard File Transfer Protocol for ARPANET, FTP acts as an interpreter on the remote machine. The user may type a number of UNIX-like commands under this interpreter to perform desired actions on the remote machine. FTP is built on a client-server architecture and utilizes separate control and data connections between the client and server. FTP users may authenticate themselves using a clear-text sign-in protocol but can connect anonymously if the server is configured to allow it.



## FTP CLIENT:

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 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.

## FTP SERVER:

The server responds on the control connection with three digit status codes in ASCII with an optional text message. The numbers represent the code number and the optional text represent explanations or needed parameters. A file transfer in progress over the data connection can be aborted using an interrupt message sent over the control connection.

## MODES:

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.

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

- ASCII mode
- Binary mode.
- EBCDIC mode.
- Local mode.

Data transfer can be done in any of three modes:

- 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.

#### **Implementation steps:**

- Configure FTP server at a particular system.
- From FTP Client program connect to FTP Server.
- After successful connection, FTP Server is ready to accept FTP command from the client.
- From FTP Client program we can send the command using socket connections, which are executed at FTP server and results are returned back to FTP Client in terms of codes.

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

Below is a **list of FTP commands** that may be sent to an FTP host, including all commands that are standardized in RFC 959 by the IETF. All commands below are RFC 959 based unless stated otherwise. These commands differ in use between clients. For example, GET is used instead of RETR, but most clients parse this into the proper command. In this, GET is the user command and RETR is the raw command.

Command	Description
ABOR	Abort an active file transfer.
ACCT	Account information.
ADAT	Authentication/Security Data ( <a href="#">RFC 2228</a> )
ALLO	Allocate sufficient disk space to receive a file.
APPE	Append.
AUTH	Authentication/Security Mechanism ( <a href="#">RFC 2228</a> )
CCC	Clear Command Channel ( <a href="#">RFC 2228</a> )
CDUP	Change to Parent Directory.
CONF	Confidentiality Protection Command ( <a href="#">RFC 697</a> )
CWD	Change working directory.
DELE	Delete file.
ENC	Privacy Protected Channel ( <a href="#">RFC 2228</a> )
EPRT	Specifies an extended address and port to which the server should connect. ( <a href="#">RFC 2428</a> )
EPSV	Enter extended passive mode. ( <a href="#">RFC 2428</a> )
FEAT	Get the feature list implemented by the server. ( <a href="#">RFC 2389</a> )
RNFR	Rename from.
RNTO	Rename to.
SITE	Sends site specific commands to remote server.
SIZE	Return the size of a file. ( <a href="#">RFC 3659</a> )
SMNT	Mount file structure.
STAT	Returns the current status.
STOR	Store (upload) a file.
STOU	Store file uniquely.
STRU	Set file transfer structure.
SYST	Return system type.
TYPE	Sets the transfer mode ( <a href="#">ASCII/Binary</a> ).
USER	Authentication username.

	is returned.
LANG	Language Negotiation ( <a href="#">RFC 2640</a> )
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. ( <a href="#">RFC 1639</a> )
LPSV	Enter long passive mode. ( <a href="#">RFC 1639</a> )
MDTM	Return the last-modified time of a specified file. ( <a href="#">RFC 3659</a> )
MIC	Integrity Protected Command ( <a href="#">RFC 2228</a> )
MKD	Make directory.
MLSD	Lists the contents of a directory if a directory is named. ( <a href="#">RFC 3659</a> )
MLST	Provides data about exactly the object named on its command line, and no others. ( <a href="#">RFC 3659</a> )
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. ( <a href="#">RFC 2389</a> )
PASS	Authentication password.
PASV	Enter passive mode.
PBSZ	Protection Buffer Size ( <a href="#">RFC 2228</a> )
PORT	Specifies an address and port to which the server should connect.
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	Retrieve (download) a remote file.
RMD	Remove a directory.
HELP	Returns usage documentation on a command if specified, else a general help document

# 1. FTP Implementation

```
//FTPSEVER.JAVA
```

```
import java.net.*;
```

```
import java.io.*;
```

```
import java.util.*;
```

```
public class FTPSERVER
```

```
{
```

```
    public static void main(String args[]) throws Exception
```

```
    {
```

```
        ServerSocket soc=new ServerSocket(5217);
```

```
        System.out.println("FTP Server Started on Port Number 5217");
```

```
        while(true)
```

```
        {
```

```
            System.out.println("Waiting for Connection ...");
```

```
            transferfile t=new transferfile(soc.accept());
```

```
        }
```

```
    }
```

```
}
```

```
class transferfile extends Thread
```

```
{
```

```
    Socket ClientSoc;
```

```
    DataInputStream din;
```

```
    DataOutputStream dout;
```

```
    transferfile(Socket soc)
```

```
    {
```

```
        try
```

```
        {
```

```
            ClientSoc=soc;
```

```
            din=new DataInputStream(ClientSoc.getInputStream());
```

```
            dout=new DataOutputStream(ClientSoc.getOutputStream());
```

```
            System.out.println("FTP Client Connected ...");
```

```
            start();
```

```
        }
```

```
        catch(Exception ex)
```

```
        {
```

```
        }
```

```
    }
```

```
void SendFile() throws Exception
```

```
{
```

```
    String filename=din.readUTF();
```

```
    File f=new File(filename);
```

```
    if(!f.exists())
```

```
    {
```

```
        dout.writeUTF("File Not Found");
```

```
        return;
```

```
    }
```

```

else
{
    dout.writeUTF("READY");
    FileInputStream fin=new FileInputStream(f);
    int ch;
    do
    {
        ch=fin.read();
        dout.writeUTF(String.valueOf(ch));
    }
    while(ch!=-1);
    fin.close();
    dout.writeUTF("File Receive Successfully");
}
}
void ReceiveFile() throws Exception
{
    String filename=din.readUTF();
    if(filename.compareTo("File not found")==0)
    {
        return;
    }
    File f=new File(filename);
    String option;

    if(f.exists())
    {
        dout.writeUTF("File Already Exists");
        option=din.readUTF();
    }
    else
    {
        dout.writeUTF("SendFile");
        option="Y";
    }

    if(option.compareTo("Y")==0)
    {
        FileOutputStream fout=new FileOutputStream(f);
        int ch;
        String temp;
        do
        {
            temp=din.readUTF();
            ch=Integer.parseInt(temp);
            if(ch!=-1)
            {
                fout.write(ch);
            }
        }
    }
}

```



```

        }while(ch!=-1);
        fout.close();
        dout.writeUTF("File Send Successfully");
    }
    else
    {
        return;
    }
}

public void run()
{
    while(true)
    {
        try
        {
            System.out.println("Waiting for Command ...");
            String Command=din.readUTF();
            if(Command.compareTo("GET")==0)
            {
                System.out.println("\tGET Command Received ...");
                SendFile();
                continue;
            }
            else if(Command.compareTo("SEND")==0)
            {
                System.out.println("\tSEND Command Receiced ...");
                ReceiveFile();
                continue;
            }
            else if(Command.compareTo("DISCONNECT")==0)
            {
                System.out.println("\tDisconnect Command Received ...");
                System.exit(1);
            }
        }
        catch(Exception ex)
        {
        }
    }
}
}

```

**//FTPCLIENT.JAVA**

```

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

```

```

class FTPCLIENT
{
    public static void main(String args[]) throws Exception
    {
        Socket soc=new Socket("127.0.0.1",5217);
        transferfileClient t=new transferfileClient(soc);
        t.displayMenu();

    }
}
class transferfileClient
{
    Socket ClientSoc;

    DataInputStream din;
    DataOutputStream dout;
    BufferedReader br;
    transferfileClient(Socket soc)
    {
        try
        {
            ClientSoc=soc;
            din=new DataInputStream(ClientSoc.getInputStream());
            dout=new DataOutputStream(ClientSoc.getOutputStream());
            br=new BufferedReader(new InputStreamReader(System.in));
        }
        catch(Exception ex)
        {
        }
    }
    void SendFile() throws Exception
    {

        String filename;
        System.out.print("Enter File Name :");
        filename=br.readLine();

        File f=new File(filename);
        if(!f.exists())
        {
            System.out.println("File not Exists...");
            dout.writeUTF("File not found");
            return;
        }

        dout.writeUTF(filename);

        String msgFromServer=din.readUTF();
        if(msgFromServer.compareTo("File Already Exists")==0)

```

```

{
    String Option;
    System.out.println("File Already Exists. Want to OverWrite (Y/N) ?");
    Option=br.readLine();
    if(Option=="Y")
    {
        dout.writeUTF("Y");
    }
    else
    {
        dout.writeUTF("N");
        return;
    }
}

System.out.println("Sending File ...");
FileInputStream fin=new FileInputStream(f);
int ch;
do
{
    ch=fin.read();
    dout.writeUTF(String.valueOf(ch));
}
while(ch!=-1);
fin.close();
System.out.println(din.readUTF());

}

void ReceiveFile() throws Exception
{
    String fileName;
    System.out.print("Enter File Name :");
    fileName=br.readLine();
    dout.writeUTF(fileName);
    String msgFromServer=din.readUTF();

    if(msgFromServer.compareTo("File Not Found")==0)
    {
        System.out.println("File not found on Server ...");
        return;
    }
    else if(msgFromServer.compareTo("READY")==0)
    {
        System.out.println("Receiving File ...");
        File f=new File(fileName);
        if(f.exists())
        {
            String Option;

```

```

System.out.println("File Already Exists. Want to OverWrite (Y/N) ?");
    Option=br.readLine();
    if(Option=="N")
    {
        dout.flush();
        return;
    }
}
FileOutputStream fout=new FileOutputStream(f);
int ch;
String temp;
do
{
    temp=din.readUTF();
    ch=Integer.parseInt(temp);
    if(ch!=-1)
    {
        fout.write(ch);
    }
}while(ch!=-1);
fout.close();
System.out.println(din.readUTF());

}

```

```

}

```

```

public void displayMenu() throws Exception
{
    while(true)
    {
        System.out.println("[ MENU ]");
        System.out.println("1. Send File");
        System.out.println("2. Receive File");
        System.out.println("3. Exit");
        System.out.print("\nEnter Choice :");
        int choice;
        choice=Integer.parseInt(br.readLine());
        if(choice==1)
        {
            dout.writeUTF("SEND");
            SendFile();
        }
        else if(choice==2)
        {
            dout.writeUTF("GET");
            ReceiveFile();
        }
    }
}

```

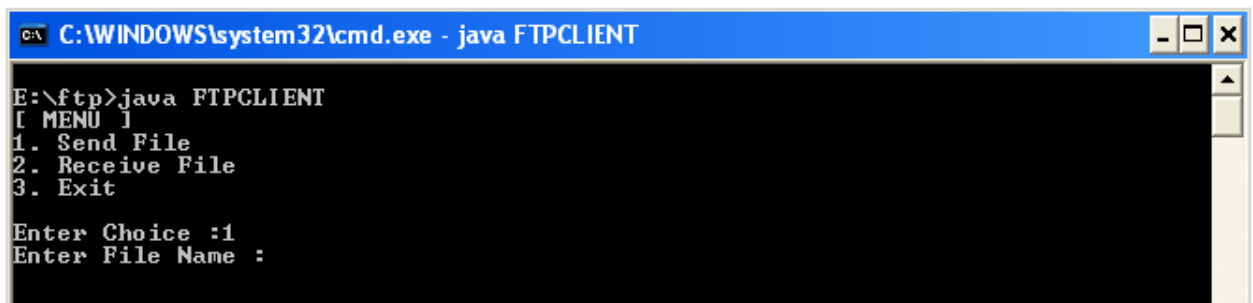
```
        else
        {
            dout.writeUTF("DISCONNECT");
            System.exit(1);
        }
    }
}
```

## FTP: OUTPUT



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

E:\ftp>javac FTPSERVER.java
E:\ftp>javac FTPCLIENT.java
E:\ftp>java FTPSERVER
FTP Server Started on Port Number 5217
Waiting for Connection ...
FTP Client Connected ...
Waiting for Connection ...
Waiting for Command ...
SEND Command Receiced ...
```



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

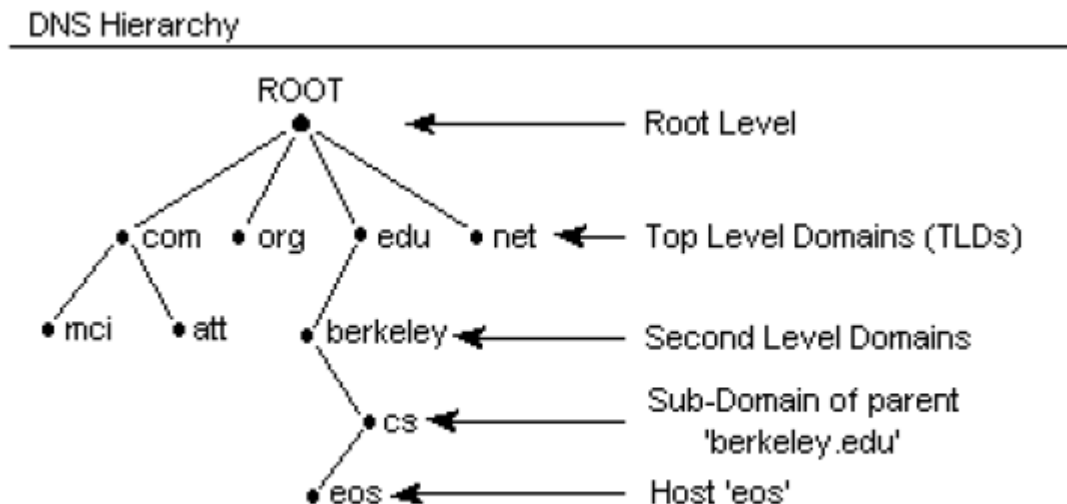
E:\ftp>java FTPCLIENT
[ MENU ]
1. Send File
2. Receive File
3. Exit

Enter Choice :1
Enter File Name :
```

## 2. Iterative Name Server Implementation Using Java

### Introduction:

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. 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. The Domain Name System distributes the responsibility of assigning domain names and mapping those names to IP addresses by designating iterative name servers for each domain. The Domain Name System is maintained by a distributed database system, which uses the client-server model. The nodes of this database are the name servers. Each domain has at least one authoritative DNS server that publishes information about that domain and the name servers of any domains subordinate to it. The top of the hierarchy is served by the root name servers, the servers to query when looking up (*resolving*) a top level domain name.



### Implementation:

- Design the table with in the database as specified in the next section.
- Implement individual server for each domain with in DNS.
- Each server program is now in waiting state.
- Implement client program which accepts URL from user, divides the URL into different domain and fetch the IPs of the domains from the corresponding servers.
- Combine the IPs received from different server and display them to the user.

## Table Name : Client

name	ipadd	port
com	10	5123
edu	20	4123
org	30	3123

## Table Name: Root

name	ipadd	port
google	13	5125
yahoo	12	5124

## Table Name: Yahoo

name	ipadd	port
mail	100	0
program	200	0

## Table Name: Google

name	ipadd	port
mail	100	0
program	200	0

### **Client.java:**

```
import java.io.*;
import java.net.*;
import java.sql.*;
import java.util.*;
class Client{
public static void main(String a[]){
Socket clisock;
DataInputStream input;
PrintStream ps;
String url,ip,s,u,p,str;
int pno=5123;
Connection con;
Statement smt;
ResultSet rs;
boolean status=true;
try{
ip=s=p=u="\0";
System.out.println("enter name to resolve");
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
url=br.readLine();
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
smt=con.createStatement();
while(status){
s="\0";
StringTokenizer st=new StringTokenizer(url,".");
if(st.countTokens()==1){status=false;}
while(st.countTokens()>1)
s=s+st.nextToken()+".";
s=s.substring(0,s.length()-1).trim();

u=st.nextToken();
rs=smt.executeQuery("select port,ipadd from client where name='"+u+"'");
if(rs.next()){
p=rs.getString(1);
pno=Integer.parseInt(p);
str=rs.getString(2);
url=s;
ip=str+"."+ip;
}
else {
clisock=new Socket("127.0.0.1",pno);
input=new DataInputStream(clisock.getInputStream());
ps=new PrintStream(clisock.getOutputStream());
ps.println(url);
p=input.readLine();
pno=Integer.parseInt(p);
```



```

str=input.readLine();
url=input.readLine();
ip=str+"."+ip;
smt.executeUpdate("insert into client values('"+u+"','"+str+"','"+p+"')");
}
ip=ip.substring(0,ip.length()-1).trim();
}
System.out.println("ip address is:"+ip);
con.close();
}
catch(Exception e)
{ System.err.println(e); }
}
}

```

### **Server1.java:**

```

import java.io.*;
import java.net.*;
import java.sql.*;
import java.util.*;
class Server1{
public static void main(String a[]){
ServerSocket sock;
Socket client;
DataInputStream input;
PrintStream ps;
String url,u,s;
Connection con;
Statement smt;
ResultSet rs;
try{
s=u+"\0";
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
smt=con.createStatement();
sock=new ServerSocket(5123);
while(true){
client=sock.accept();
input=new DataInputStream(client.getInputStream());
ps=new PrintStream(client.getOutputStream());
url=input.readLine();
StringTokenizer st=new StringTokenizer(url,".");
while(st.countTokens()>1)
s=s+st.nextToken()+".";
s=s.substring(0,s.length()-1).trim();
u=st.nextToken();
rs=smt.executeQuery("select port,ipadd from root where name='"+u+"'");
if(rs.next()){

```

```

ps.println(Integer.parseInt(rs.getString(1)));
ps.println(Integer.parseInt(rs.getString(2)));
ps.println(s);
}
else{
ps.println("Illegal address pleasr check the spelling again");
con.close();
}
}
}
catch(Exception e)
{ System.err.println(e); }
}
}

```

### **Server2.java:**

```

import java.io.*;
import java.net.*;
import java.sql.*;
import java.util.*;
class Server2{
public static void main(String a[]){
ServerSocket sock;
Socket client;
DataInputStream input;
PrintStream ps;
String url,u,s;
Connection con;
Statement smt;
ResultSet rs;
try{
s=u="\0";
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
smt=con.createStatement();
sock=new ServerSocket(5124);
while(true){
client=sock.accept();
input=new DataInputStream(client.getInputStream());
ps=new PrintStream(client.getOutputStream());
url=input.readLine();
StringTokenizer st=new StringTokenizer(url,".");
while(st.countTokens()>1)
s=s+st.nextToken()+ ".";
s=s.substring(0,s.length()-1).trim();
u=st.nextToken();
rs=smt.executeQuery("select port,ipadd from yahoo where name='"+u+"'");
if(rs.next()){

```

```

ps.println(rs.getString(1));
ps.println(rs.getString(2));
ps.println(s);
}
else{
ps.println("Illegal address pleasr check the spelling again");
con.close();
}
}
}
catch(Exception e)
{ System.err.println(e); }
}
}

```

### **Server3.java:**

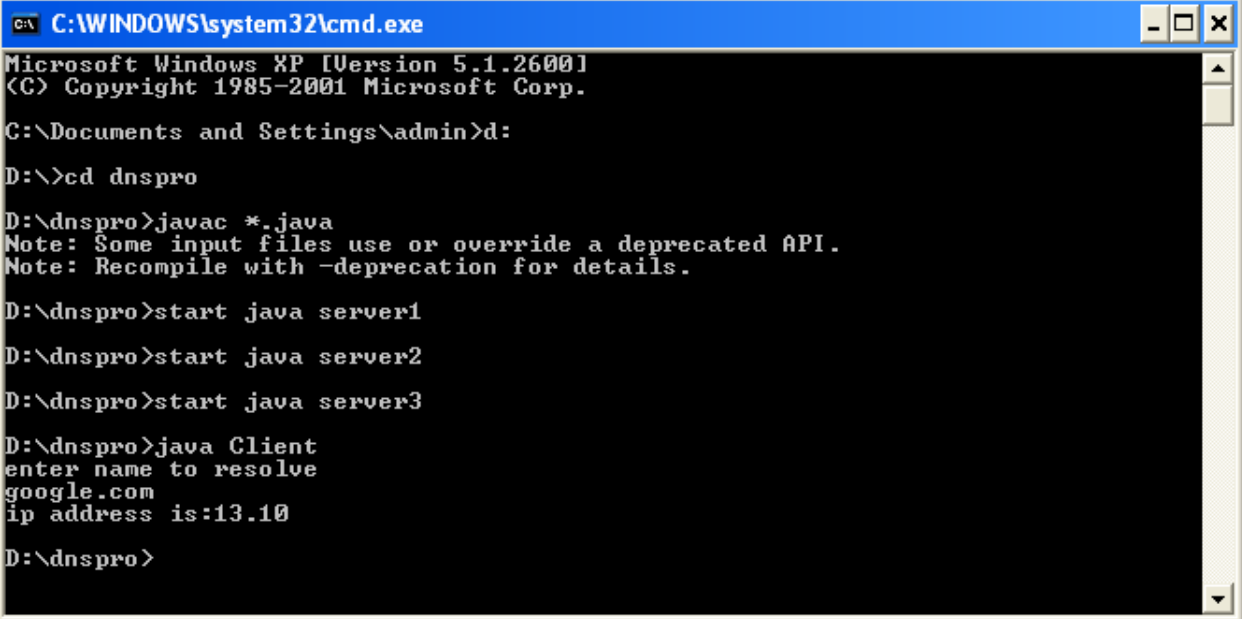
```

import java.io.*;
import java.net.*;
import java.sql.*;
import java.util.*;
class Server3{
public static void main(String a[]){
ServerSocket sock;
Socket client;
DataInputStream input;
PrintStream ps;
String url,u,s;
Connection con;
Statement smt;
ResultSet rs;
try{
s=u+"\0";
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
smt=con.createStatement();
sock=new ServerSocket(5125);
while(true){
client=sock.accept();
input=new DataInputStream(client.getInputStream());
ps=new PrintStream(client.getOutputStream());
url=input.readLine();
StringTokenizer st=new StringTokenizer(url,".");
while(st.countTokens()>1)
s=s+st.nextToken()+ ".";
s=s.substring(0,s.length()-1).trim();
u=st.nextToken();
rs=smt.executeQuery("select port,ipadd from google where name='"+u+"'");
if(rs.next()){

```

```
ps.println(rs.getString(1));
ps.println(rs.getString(2));
ps.println(s);
}
else{
ps.println("Illegal address please check the spelling again");
con.close();
}
}
}
catch(Exception e)
{ System.err.println(e); }
}
}
```

### Name Server: OUTPUT



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

C:\Documents and Settings\admin>d:
D:\>cd dnspro
D:\dnspro>javac *.java
Note: Some input files use or override a deprecated API.
Note: Recompile with -deprecation for details.
D:\dnspro>start java server1
D:\dnspro>start java server2
D:\dnspro>start java server3
D:\dnspro>java Client
enter name to resolve
google.com
ip address is:13.10
D:\dnspro>
```

### 3. Iterative Name Server Implementation Using JSP

#### Main.jsp:

```
<form method = "post" action = "Dns.jsp" target = "_blank">
<input type = "text" name = "dn" />
<input type = "button" value = "Search on DNS" onclick = "submit();
self.close();" />
</form>
```

#### Dns.jsp:

```
<% @ page import="java.sql.*" %>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String dn = request.getParameter("dn");
try{
Connection con =
DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
PreparedStatement ps = con.prepareStatement("select addr from dns where dn='"+dn+"'");
ResultSet rs = ps.executeQuery();
rs.next();
String addr = rs.getString(1);
%>
<form method = "post" action = "<%=addr%>" target = "_blank">
<input type = "button" value = "Go to Site" onclick = "submit()" />
</form>
<%
ps.close();
}
catch(SQLException e)
{
%>Domain address Not available<%
}%>
```

#### Google.jsp:

Welcome to Google

#### Yahoo.jsp:

Welcome to Yahoo

#### Facebook.jsp:

Welcome to Facebook

#### Output:

##### Procedure to create and insert values into table dns :

```
SQL> create table dns(dn varchar2(20), addr varchar2(50));
```

Table created.

```
SQL> insert into dns values('google.com','http://localhost/Progs/NameServer/google.jsp');
```

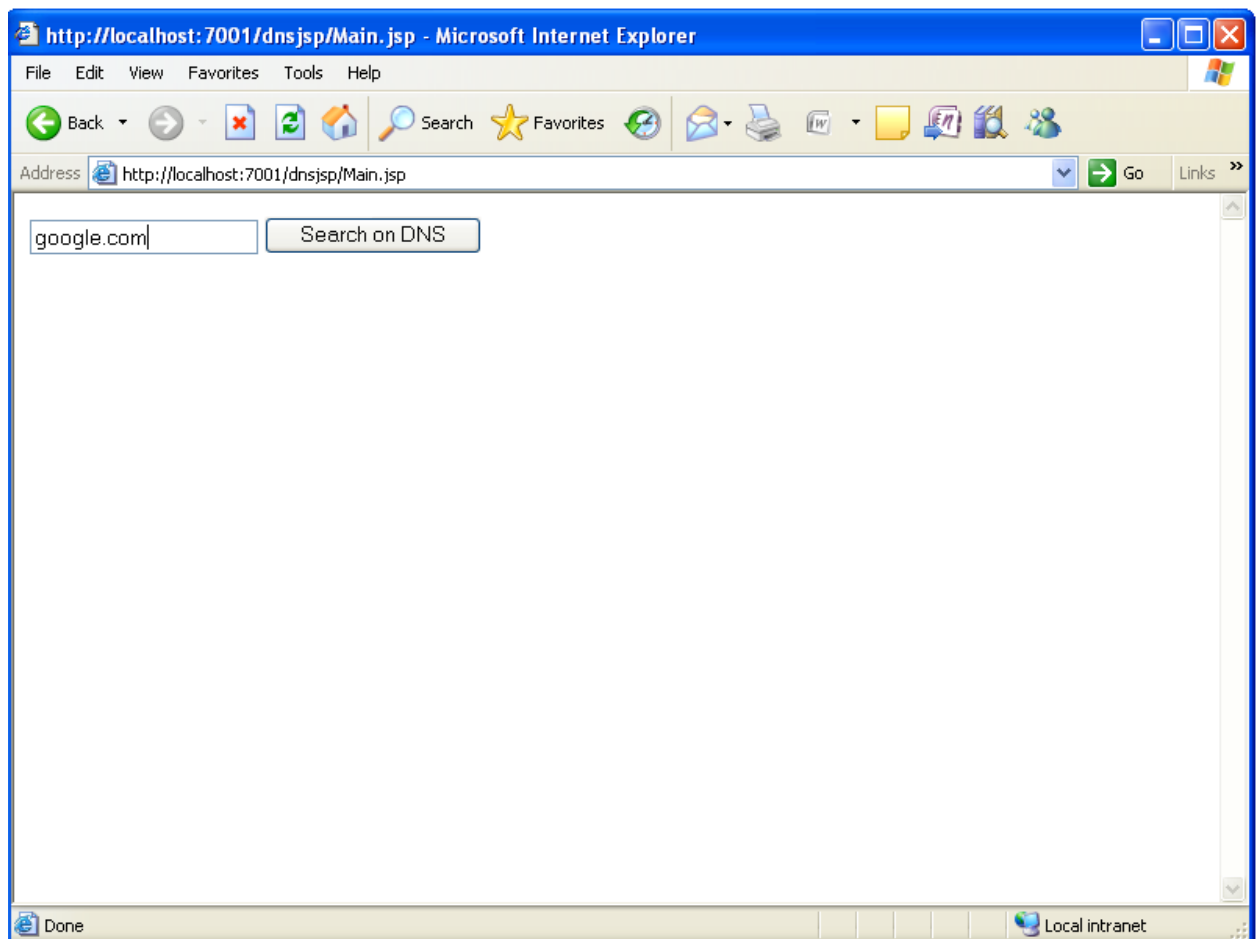
1 row created.

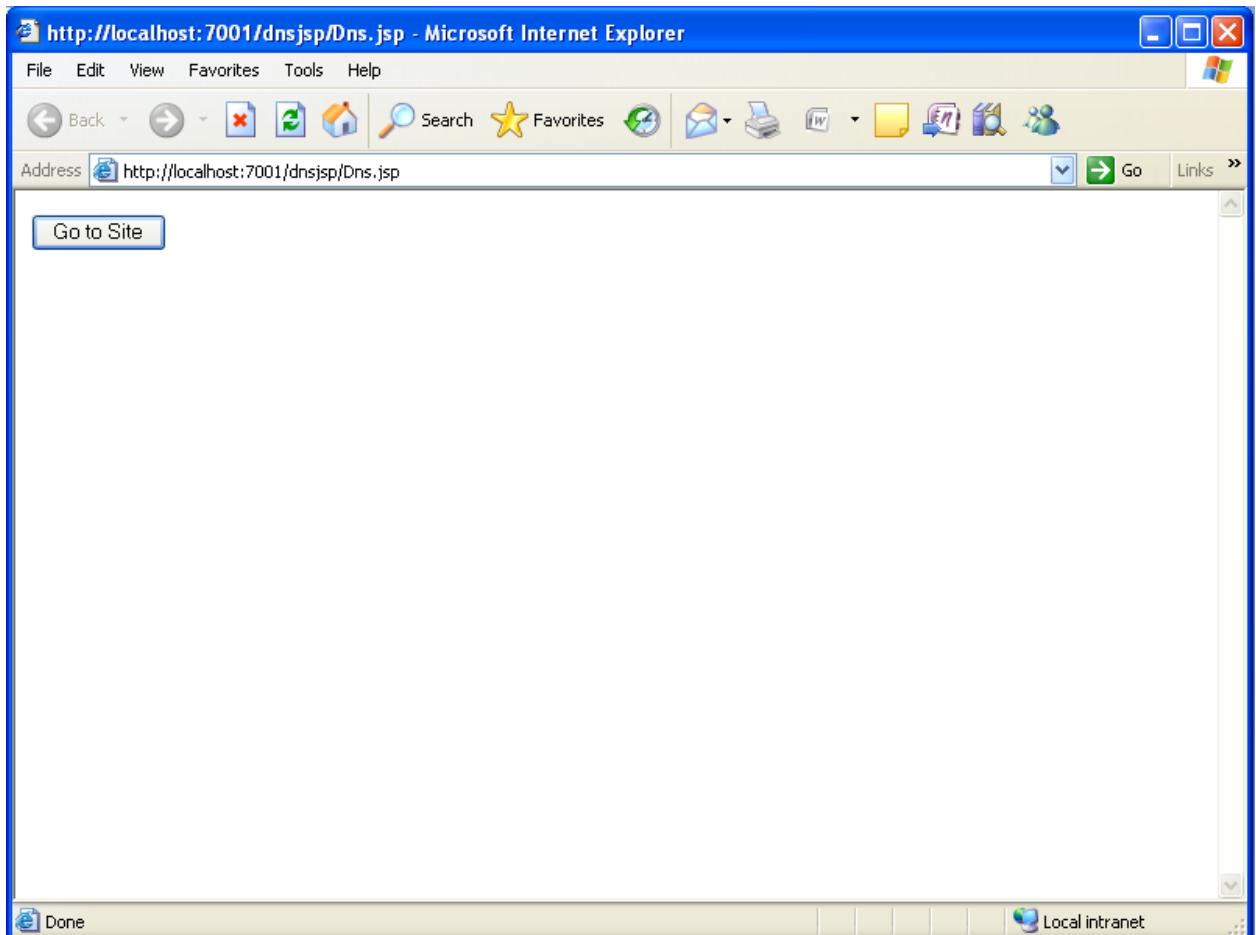
```
SQL> insert into dns values('yahoo.com','http://localhost/Progs/NameServer/yahoo.jsp');
1 row created.
```

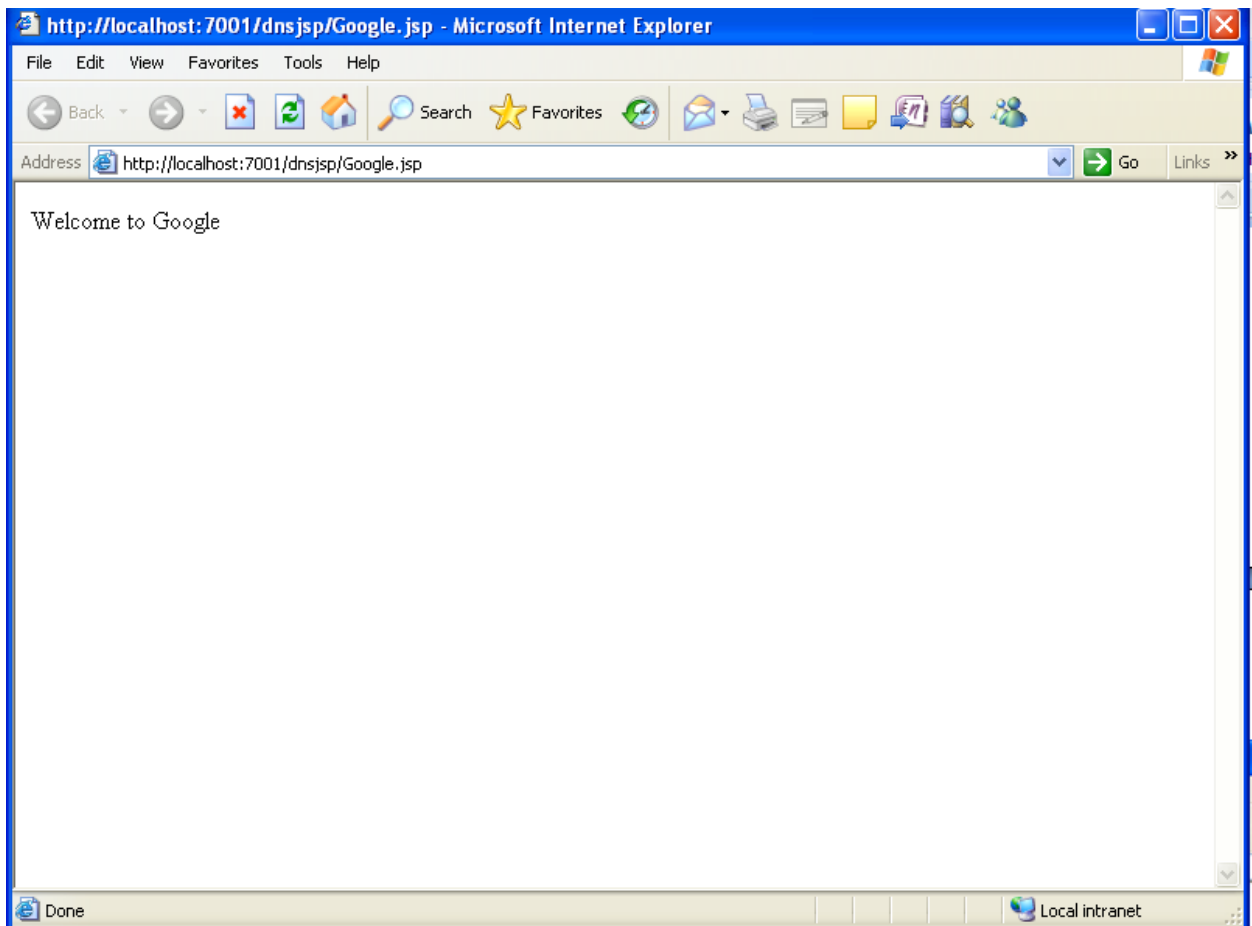
```
SQL> insert into dns
values('facebook.com','http://localhost/Progs/NameServer/facebook.jsp');
1 row created.
```

```
SQL> select * from dns;
DN ADDR
```

```
-----
google.com http://localhost/Progs/NameServer/google.jsp
yahoo.com http://localhost/Progs/NameServer/yahoo.jsp
facebook.com http://localhost/Progs/NameServer/facebook.jsp
SQL> commit;
Commit complete.
```









## 4. Chat Server Implementation

### Introduction:

Chat server is a stand alone application that is made of combination of two-applications, 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 client must be connected with the server after which, messages can be broadcast between each and every client.

### SERVER:

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.

- Firstly server program creates a new server socket by the **ServerSocket ss = new ServerSocket(Port number);**
- After creating the ServerSocket it accepts the client socket and adds this socket into an arraylist.
- After getting the client socket it creates a thread and enables the DataInputStream for this socket.
- After creating the input stream it reads the user name and adds it to arraylist and this arraylist object writes into the ObjectOutputStream of each client by using an iterator.
- After this process, it creates a new thread and creates one DataInputStream for reading the messages which is sent by the client and after reading the message it creates the DataOutputStream for each socket and writes this message in each client output stream through the iterator. If any client logs out,the server receives the client name and removes it from the arraylist. Further ,it sends this updated arraylist to all clients.

### CLIENT:

In the client side, first the program creates a new socket and specifies the address and port of the server and establishes the connection with the Server.The client, then, creates a new thread and DataInputStream, ObjectInputStream and DataOutputStream for sending the user name and retrieving the list of all users and adds all the user names into its list box through the iterator.Then new thread is created for sending and receiving the messages from the server. This task is done by using DataInputStream and DataOutputStream.When the client logs out it sends its' name and message i.e. "User\_Name has Logged out" and terminates the chatting.

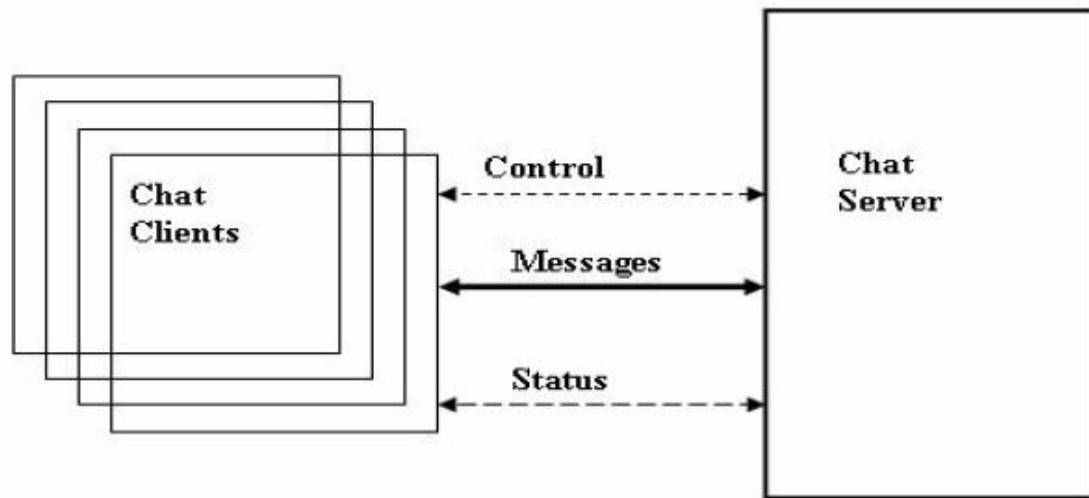


Figure 1: Chat Application Overview

### ChatServer.java:

```

import java.net.*;
import java.io.*;
public class ChatServer implements Runnable {
    private ChatServerThread clients[]=new ChatServerThread[50];
    private ServerSocket server=null;
    private Thread thread=null;
    private int clientCount=0;
    public ChatServer(int port) {
        try {
            System.out.println("binding to port"+port+",please wait ...");
            server=new ServerSocket(port);
            System.out.println("server started:"+server);
            thread=new Thread(this);
            thread.start();
        }
        catch(IOException e)
        { System.out.println("cannot bind to port"+port+"."+e.getMessage()); }
    }
    public void run() {
        while(thread!=null) {
            try {
                System.out.println("waiting for a client...");
                addThread(server.accept());
            }
            catch(IOException e)
            {
                System.out.println("server accept error:"+e);
                if(thread!=null)
                {

```

```

thread.stop();
thread=null;
}
}
}
}
public void stop()
{
if(thread!=null)
{
thread.stop();
thread=null;
}
}
private int findClient(int ID) {
for(int i=0;i<clientCount;i++)
if(clients[i].getID()==ID)
return i;
return -1;
}
public synchronized void handle(int ID,String input) {
if(input.equals("quit"))
{
clients[findClient(ID)].send("quit");
remove(ID);
}
else
System.out.println(ID+": "+input);
for(int i=0;i<clientCount;i++)
clients[i].send(ID+": "+input);
}
public synchronized void remove(int ID) {
int pos=findClient(ID);
if(pos>=0) {
ChatServerThread closing=clients[pos];
System.out.println("removing client thread:"+ID+"at"+pos);
if(pos<clientCount-1)
for(int i=pos+1;i<clientCount;i++)
clients[i-1]=clients[i];
clientCount--;
try {
closing.close();
}
catch(IOException e)
{ System.out.println("error closing thread;" +e); }
closing.stop();
}
}
private void addThread(Socket socket) {

```

```

if(clientCount<clients.length) {
System.out.println("client accepted:"+socket);
clients[clientCount]=new ChatServerThread(this,socket);
try {
clients[clientCount].open();
clients[clientCount].start();
clientCount++;
}
catch(IOException e)
{ System.out.println("error opening thread;" +e); }
}
else
System.out.println("client refused;maximum"+clients.length+"reached");
}
public static void main(String a[])
{
ChatServer server=null;
if(a.length!=1)
System.out.println("Usage:java ChatServer Port");
else
server=new ChatServer(Integer.parseInt(a[0]));
}
}

```

### **ChatServerThread.java:**

```

import java.net.*;
import java.io.*;
public class ChatServerThread extends Thread {
private ChatServer server=null;
private Socket socket=null;
private DataInputStream In=null;
private int ID=-1;
private PrintStream Out=null;
public ChatServerThread(ChatServer serv,Socket sock) {
super();
server=serv;
socket=sock;
ID=socket.getPort();
}
public void send(String msg) {
Out.println(msg);
Out.flush();
}
public int getID()
{
return ID;
}
public void run() {

```

```

System.out.println("Server thread"+ID+"running");
while(true) {
try {
server.handle(ID,In.readLine());
}
catch(IOException e) {
System.out.println(ID+"error reading"+e.getMessage());
server.remove(ID);
stop();
}
}
}
}
public void open()throws IOException
{
In=new DataInputStream(socket.getInputStream());
Out=new PrintStream(socket.getOutputStream());
}
public void close()throws IOException
{
if(socket!=null)
socket.close();
if(In!=null)
In.close();
if(Out!=null)
Out.close();
}
}
}

```

### **ChatClient.java:**

```

import java.net.*;
import java.io.*;
public class ChatClient implements Runnable {
private ChatClientThread client=null;
private Socket socket=null;
private DataInputStream console=null;
private Thread thread=null;
private PrintStream Out=null;
public ChatClient(String serverName,int serverPort) {
System.out.println("Establishing connection please wait...");
try {
socket=new Socket(serverName,serverPort);
System.out.println("connected"+socket.toString());
console=new DataInputStream (System.in);
Out=new PrintStream(socket.getOutputStream());
if(thread==null)
{
client=new ChatClientThread(this,socket);
thread=new Thread(this);
}
}
}
}

```

```

thread.start();
}
}
catch(UnknownHostException e)
{
System.out.println("Host unknown"+e.getMessage());
}
catch(IOException ioe)
{
System.out.println("Unexcepected exception"+ioe.getMessage());
}
}
public void run() {
while(thread!=null) {
try {
Out.println(console.readLine());
Out.flush();
}
catch(IOException e)
{
System.out.println("sending error"+e.getMessage());
stop();
}
}
}
public void handle(String msg) {
if(msg.equals("quit"))
{
System.out.println("good bye, press RETURN to exit...");
stop();
}
else
System.out.println(msg);
}
public void stop() {
if(thread!=null) {
thread.stop();
thread=null;
}
try {
if(console!=null)
console.close();
if(Out!=null)
Out.close();
if(socket!=null)
socket.close();
}
catch(IOException e)
{

```

```

System.out.println("error closing...");
}
client.close();
client.stop();
}
public static void main(String args[]) {
ChatClient client=null;
if(args.length!=2)
System.out.println("usage:java ChatClient host port");
else
client=new ChatClient(args[0],Integer.parseInt(args[1]));
}
}

```

### **ChatClientThread.java:**

```

import java.net.*;
import java.io.*;
public class ChatClientThread extends Thread {
private ChatClient client=null;
private Socket socket=null;
private DataInputStream In=null;
public ChatClientThread(ChatClient cli,Socket sock)
{
client=cli;
socket=sock;
try
{
In=new DataInputStream(socket.getInputStream());
}
catch (IOException e)
{
System.out.println("error geting input stream:"+e);
client.stop();
}
start();
}
public void close()
{
try
{
if(In!=null)
In.close();
}
catch (IOException e)
{
System.out.println("error closing input stream:"+e);
}
}
public void run()

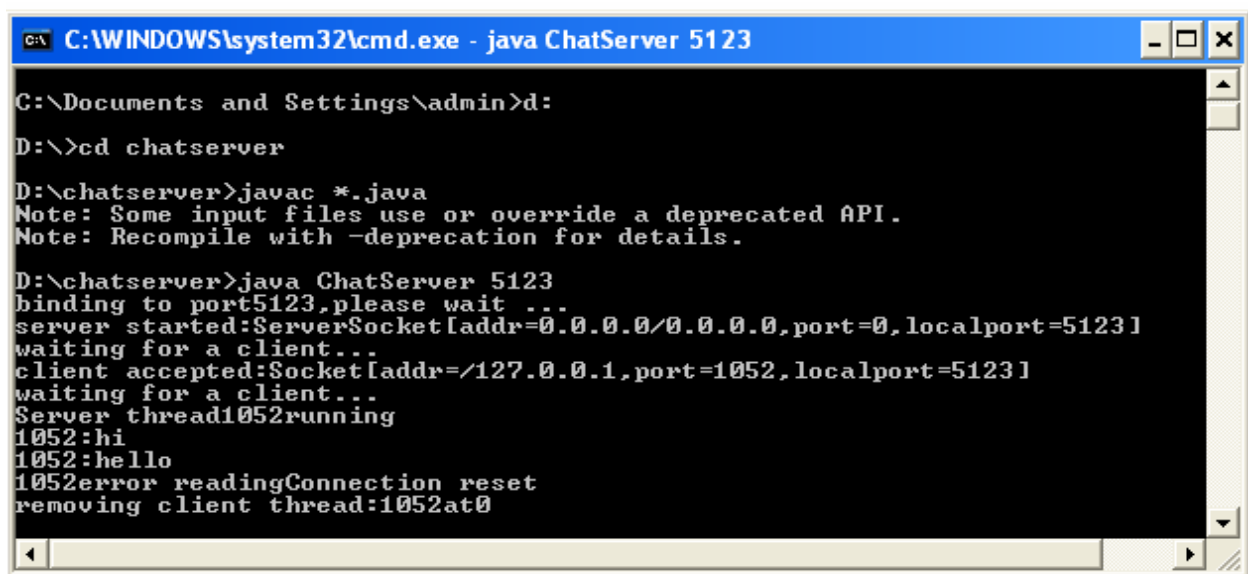
```

```

{
while(true)
{
try
{
client.handle(In.readLine());
}
catch(IOException e)
{
System.out.println("Listening error"+e.getMessage());
client.stop();
}
}
}
}
}

```

### ChatServer: OUTPUT



```

C:\WINDOWS\system32\cmd.exe - java ChatServer 5123
C:\Documents and Settings\admin>d:
D:\>cd chatserver
D:\chatserver>javac *.java
Note: Some input files use or override a deprecated API.
Note: Recompile with -deprecation for details.
D:\chatserver>java ChatServer 5123
binding to port5123,please wait ...
server started:ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=5123]
waiting for a client...
client accepted:Socket[addr=/127.0.0.1,port=1052,localport=5123]
waiting for a client...
Server thread1052running
1052:hi
1052:hello
1052error readingConnection reset
removing client thread:1052at0

```



```
C:\WINDOWS\system32\cmd.exe - java ChatClient sys1-1a589e1837 5123
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\admin>d:

D:\>cd chatserver

D:\chatserver>java ChatClient sys1-1a589e1837 5123
Establishing connection please wait...
connectedSocket[addr=sys1-1a589e1837/127.0.0.1,port=5123,localport=1052]
hi
1052:hi
hello
1052:hello
```

## 5. Chat Server Implementation Using JSP

### Start.jsp:

```
<h2>Chat Server</h2>
<form method="post" action="Login.jsp" >
UserName : <input type="text" name="usr" /><br><br>
Password : <input type="password" name="pass" /><br><br>
<input type="button" value="Log In" onclick="submit()"/>
</form>
```

### Login.jsp:

```
<% @ page import="java.sql.*" %>
<%
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
String usr=request.getParameter("usr");
String pass=request.getParameter("pass");
try
{
Connection con
=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
PreparedStatement stmt=con.prepareStatement("select * from login where usr='"+usr+"'
and pass='"+pass+"'");
ResultSet rs=stmt.executeQuery();
if(rs.next())
{ %>
<form method="post" action="Cboxs.jsp">
<input type="hidden" name="flag" value="<%=usr%>"/>
<input type="button" value="Go to CHAT BOX - - >" onclick="submit()"/>
</form>
<% }
else
{
%>
Login failed<br>
<form method="post" action="Strt.jsp" >
<input type="button" value="< - - Go Back" onclick="submit()"/>
</form>
<%
}
stmt.close();
}
catch(SQLException se)
{
%>
Login ERROR<br>
<form method="post" action="Start.jsp" >
```

```

<input type="button" value="< - Go Back" onclick="submit()"/>
</form>
<%
}
%>

```

### **Cboxs.jsp:**

```

<% @ page import="java.sql.*" %>
<%
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
String flag=request.getParameter("flag");
try
{
Connection con
=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
PreparedStatement stmt=con.prepareStatement("select * from msg where
uto='"+flag+"'");
ResultSet rs=stmt.executeQuery();
%>
<form method="post" action="Writes.jsp" >
To : <select name="uto">
<option value="abc"/>abc
<option value="xyz"/>xyz
<option value="pqr"/>pqr
<option value="mno"/>mno
</select>Message : <input type="text" name="msg" />
<input type="hidden" name="flag" value="<%=flag%>" />
<input type="button" value="Send Message" onclick="submit()"/>
</form>
<form method="post" action="Start.jsp" >
<input type="button" value="LogOut" onclick="submit()"/>
</form>
<form method="post" action="Cboxs.jsp" >
<input type="hidden" name="flag" value="<%=flag%>" />
<input type="button" value="Refresh Messages" onclick="submit()"/>
</form>
<h3>Messages</h3><%
while(rs.next())
{
out.println(rs.getString(1)+" : "+rs.getString(3)+"<br>");
}
stmt.close();

}
catch(SQLException se)
{
%>
Database ERROR

```

```
<%  
}  
%>
```

### **Writes.jsp:**

```
<% @ page import="java.sql.*" %>  
<%  
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");  
String flag=request.getParameter("flag");  
String msg=request.getParameter("msg");  
String uto=request.getParameter("uto");  
try  
{  
Connection con  
=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");  
PreparedStatement stmt=con.prepareStatement("insert into msg  
values '"+flag+"','"+uto+"','"+msg+"'");  
int rs=stmt.executeUpdate();  
if(rs == 1)  
out.println("written successfully");  
%>  
<br><form method="post" action="Cboxs.jsp" >  
<input type="hidden" name="flag" value="<%=flag%>" />  
<input type="button" value="< - - Go back to CHATBOX" onclick="submit()" />  
</form>  
<%  
stmt.close();  
}  
catch(SQLException se)  
{  
%>  
Database ERROR  
<%  
}  
%>
```

## Output:

### Procedure to create and insert values into table login:

```
SQL> create table login (usr varchar2(20),pass varchar2(20));
```

Table created.

```
SQL> insert into login values('abc','abc');
```

1 row created.

```
SQL> insert into login values('xyz','xyz');
```

1 row created.

```
SQL> commit;
```

Commit complete.

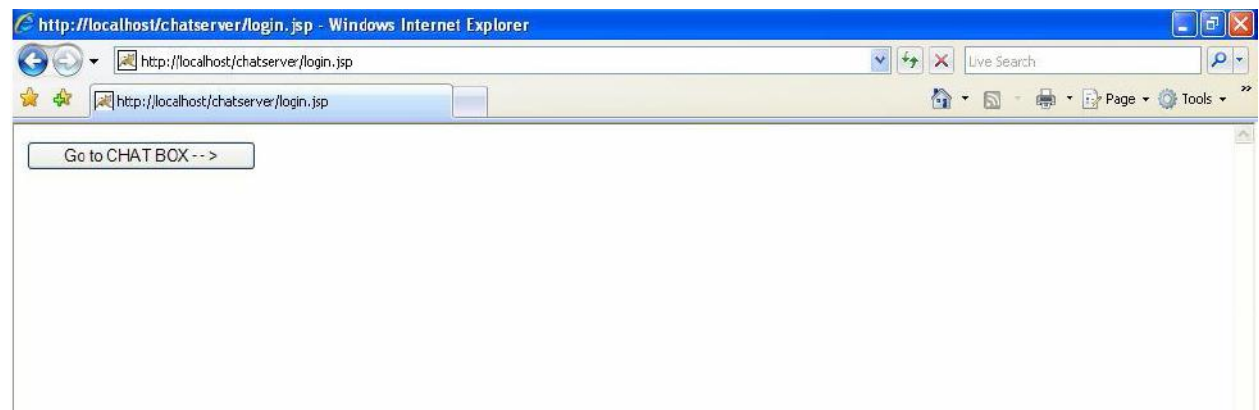
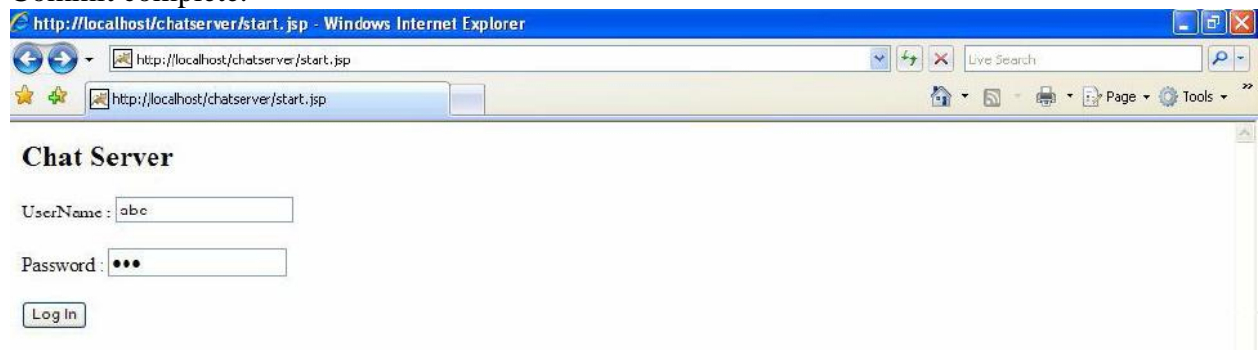
### Procedure to create and insert values into table msg:

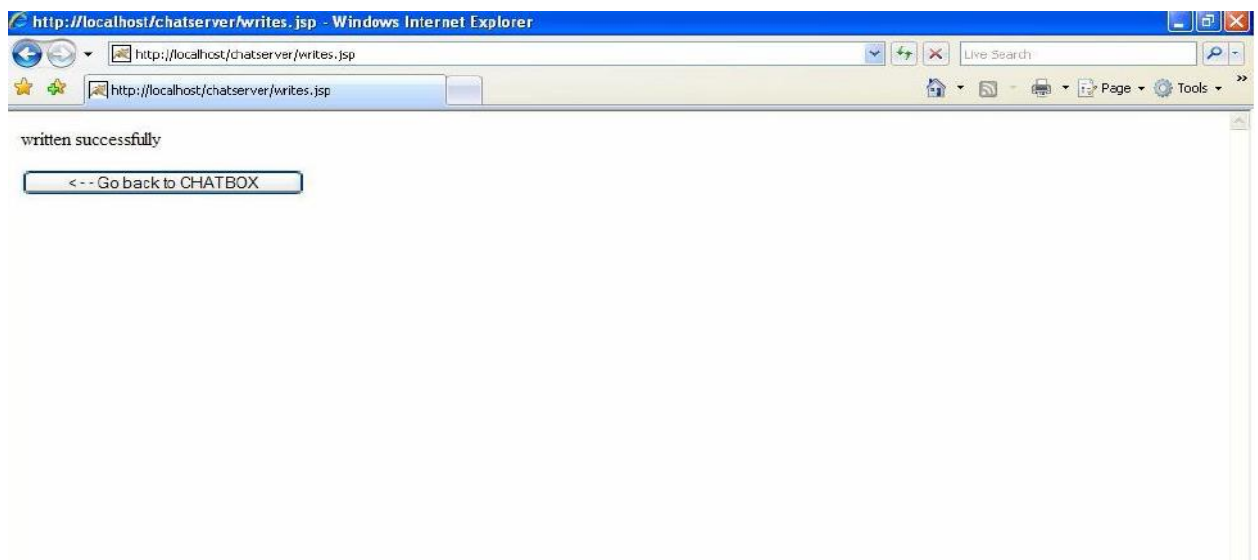
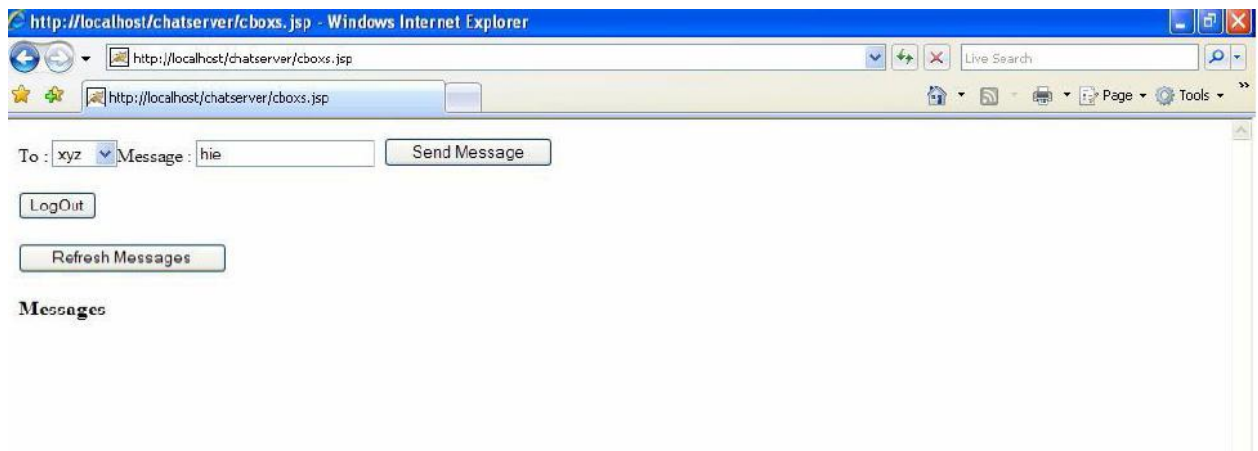
```
SQL> create table msg(ufrom varchar2(20),uto varchar2(20),msg varchar2(30));
```

Table created.

```
SQL> commit;
```

Commit complete.





**Output in msg table:**

SQL> select \* from msg;

UFROM	UTO	MSG
abc	xyz	hi

## 6. Understanding Working of NFS

### Servers:

File sharing Server's:

1. NFS: Network File System

These servers are used in LAN

2. FTP: File Transfer Protocol

These servers are used in WAN & LAN

3. Samba Server:

These servers are used in Linux & Window heterogeneous networks.

Note: 1. NFS is used in LAN because bandwidth for NFS is more than compared with FTP and

Samba.

2. STP is used in WAN because FTP requires very less bandwidth when compared to

NFS.

### NFS (Network File System):-

#### Configuration Steps:

- 1) Copy all the shared files to a single location

/nfsshare

aa bb cc

- 2) Create new file inside the share directory

aa bb cc

- 3) Packages:

i) nfs-utils: to un-install the nfs-utils

ii) portmap (please do not uninstall portmap)

- 4) services nfs

portmap

- 5) Daemons:

nfsd

quotad

mountd

- 6) Main Configuration File: /etc/exports

#### Experiment:

- Make a directory named nfsshare with files aa bb cc:

```
#mkdir /nfsshare
```

```
#cd /nfsshare
```

```
#touch aa bb cc
```

```
#ls
```

```
aa bb cc
```

- Check if the nfs-utils package is installed or not:

```
#rpm -q nfs-utils (Prints a message whether package is installed or not)
```

```
#rpm -q portmap
```

- To reinstall the package first remove it with the following commands:

```
#services nfs stop
```

```
#rpm -e nfs-utils
#rm -rf /var/lib/nfs/xtab----- remove
```

➤ If package is not installed then there are two ways to install:

1. Download from FTP and install

```
#ping the server
#rpm -ivh ftp://192.168.0.250:/pub.RedHat/RPMS/nfs* --force -aid
```

2. Install from CD

```
#mount /dev/cdrom/mnt
#cd /mnt
#ls
#cd Fedora
#cd RPMS
#rpm -ivh nfs-utils* --force -aid
#rpm -ivh portmap* --force -aid
```

➤ After installing the nfs-utils package create file as below:

```
#vi /etc/exports
/var/ftp/pub 192.168.0.0./24(ro,sync)
/nfsshare 192.168.0.0.24(rw,sync)
```

Note: in vi-editor write this content (/nfsshare server ipaddress and no. of systems that are connected in network)

➤ After installing services enter the command to restart

```
#services nfs restart
```

Note: execute this command twice because first it will show failed second time it will show

ok.

➤ Access to NFS share from client:

```
#mount -t nfs 10.10.12.114:/nfsshare/mnt
```

Note: In client machine enter Server Ipaddress

Note:

- 1) #ping 192.168.0.3 -b Broadcasts the address in the network only.
- 2) #ssh 192.168.0.8 Connects the PC to another PC just like Terminal connection in Windows.
- 3) In NFS all files & Directory are by default in read only mode.

Common KVM Switch: Using a KVM switch a monitor, keyboard, and a mouse can be connected to two computers.



## 7. Bulletin Board – Implementation

### Introduction:

Bulletin Board application provides group communication between different group members. Here we are maintaining different groups, and each group is having two or more members. Within a particular group any member can post the message which can be read by all the group members and can read the message posted by other group members. Here we are maintaining separate message table for every group.

### Implementation:

- Create the database and tables as shown below.
- Establish the connection to the database using JDBC from the program.
- Insert all the messages posted by particular group members into its corresponding table.
- If any group member wants to read the message, extract all messages from its group table and display them.

Create a database called “BULLETIN” and then create the following tables in it:

Table Name : **Group1MEM**

Member	passwd
abc	abc123
xyz	xyz123

Table Name : **Groups** (GrpName+“Mem”)

Group	Total
Group1	3
Group2	2

Table Name : **Group2Mem**

Member	passwd
pqr	pqr123
lmn	lmn123

Table Name : **Group1Msg** (GrpName+“Msg”)

Message	user
Hello	abc
Hi all!	xyz

Table Name : **Group2Msg**

Message	user
Hi all!	pqr
This is LMN	lmn

**bulletin.java**

```

import java.io.*;
import java.sql.*;
public class bulletin {
static Connection con;
static Statement st;
static ResultSet rs;
public static void main(String[] args)throws IOException {
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
st=con.createStatement();
}
catch(ClassNotFoundException ex) {
ex.printStackTrace();
}
catch(SQLException ex) {
ex.printStackTrace();
}
if(con!=null) {
System.out.println("Connected to server...\n");
System.out.println("Please Enter your group name: \n");
try {
rs=st.executeQuery("Select * from groups");
while(rs.next()) {
System.out.println(rs.getString("Grp"));
}
}
catch(SQLException ex) {

ex.printStackTrace();
}
String grpname;
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
grpname=br.readLine();
System.out.println("Enter username: \t");
String usr=br.readLine();
System.out.println("\nEnter password: \t");
String pass=br.readLine();
try {
String query="select * from "+grpname.trim()+"Mem where Member='"+usr+"'";
System.out.println(query);
rs=st.executeQuery(query);
while(rs.next()) {
if(rs.getString("passwd").equals(pass)) {
System.out.println("Login Successful!!!HAVE A NICE DAY!!!");
while(true) {
System.out.println("Enter your choice: \n");
System.out.println("1. Read messages \n");

```

```

System.out.println("2. Write messages \n");
System.out.println("3. Logout \n");
int option=Integer.parseInt(br.readLine());
switch(option) {
case 1:
query="select * from "+grpname+ "Msg";
System.out.println(query);
rs=st.executeQuery(query);
while(rs.next())
System.out.println(rs.getString("usr")+":"+rs.getString("Message")+"\n\n");
break;
case 2:

System.out.println("Enter your message: \n");
String msg=br.readLine();
query="insert into "+grpname+"Msg"+" values('"+msg+"','"+usr+"')";
System.out.println(query);
st.executeUpdate(query);
break;
case 3:
System.exit(1);
default:
System.out.println("Choose a valid option!!! \n");
break;
}
}
}
else {
System.out.println("Please login again!!!\n");
System.exit(1);
}
}
}
catch(SQLException ex)
{ ex.printStackTrace(); }
}
}
}

```

## **Bulletin Board: OUTPUT**

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

C:\Documents and Settings\admin>d:
D:\>cd 3104
D:\3104>javac *.java
Note: ftp.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
D:\3104>java bulletin
Connected to server...

Please Enter your group name:
group1
group2
group1
Enter username:
abc

Enter password:
abc123
select * from group1Mem where Member='abc'
Login Successful!!!HAVE A NICE DAY!!!
Enter your choice:
1. Read messages
2. Write messages
3. Logout
1
select * from group1Msg
abc:Hello

abc:hello hai

abc:xgdhfgjgvjhbbj

Enter your choice:
1. Read messages
2. Write messages
3. Logout
```

```
select * from Group1Msg
abc:Hello
xyz:Hi all!
```

## 8. Bulletin Board Implementation Using JSP

### Start.jsp:

```
<h2>Bulletin Board</h2>
<form method="post" action="bb.jsp" >
Select Group : <select name="grp">
<option value="grp1"/>Group-1
<option value="grp2"/>Group-2
</select><br><br>
UserName : <input type="text" name="usr" /><br><br>
Password : <input type="password" name="pass" /><br><br>
<input type="button" value="Log In" onclick="submit()"/>
</form>
```

### bb.jsp:

```
<% @ page import="java.sql.*" %>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String flag=request.getParameter("flag");
String msg=request.getParameter("msg");
try
{
Connection con
=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
PreparedStatement stmt=con.prepareStatement("insert into "+flag+" values '"+msg+"'");
int rs=stmt.executeUpdate();
if(rs == 1)
out.println("written successfully");
%>
<br><form method="post" action="Inbox.jsp" >
<input type="hidden" name="flag" value="<%=flag%>"/>
<input type="button" value="< - - Go back to INBOX" onclick="submit()"/>
</form>
<%
stmt.close();
}
catch(SQLException se)
{
%>
Database ERROR
<%
}
%>
```

**Inbox.jsp:**

```

<%
String flag=request.getParameter("flag");
%>
<h2>INBOX</h2>
<form method="post" action="read.jsp" target="_blank">
<input type="hidden" name="flag" value="<%=flag%>" />
<input type="button" value="Read Messages" onclick="submit()" />
</form><br>
<form method="post" action="write.jsp" target="_blank">
<input type="hidden" name="flag" value="<%=flag%>" />
<input type="button" value="Write Message" onclick="submit()" />
</form><br>
<form method="post" action="Start.jsp" >
<input type="hidden" name="flag" value="<%=flag%>" />
<input type="button" value="LogOut" onclick="submit()" />
</form>

```

**Read.jsp:**

```

<% @ page import="java.sql.*" %>
<%
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
String flag=request.getParameter("flag");
try
{
Connection con
=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");
PreparedStatement stmt=con.prepareStatement("select * from "+flag+"");
ResultSet rs=stmt.executeQuery();
%><h3>Messages</h3><br><%
while(rs.next())
{
out.println(rs.getString(1)+"<br>");
}
%>
<br><form method="post" action="Inbox.jsp" >
<input type="hidden" name="flag" value="<%=flag%>" />
<input type="button" value="< - Go back to INBOX" onclick="submit()" />
</form>
<%
stmt.close();
}
catch(SQLException se)
{
%>
Database ERROR
<%
}

```

%>

### **Write.jsp:**

```
<%  
String flag=request.getParameter("flag");  
%>  
<form method="post" action="update.jsp" >  
<input type="hidden" name="flag" value="<%=flag%>" />  
Enter message here : <input type="text" name="msg" /><br>  
<input type="button" value="Write message to Database" onclick="submit()" />  
</form>
```

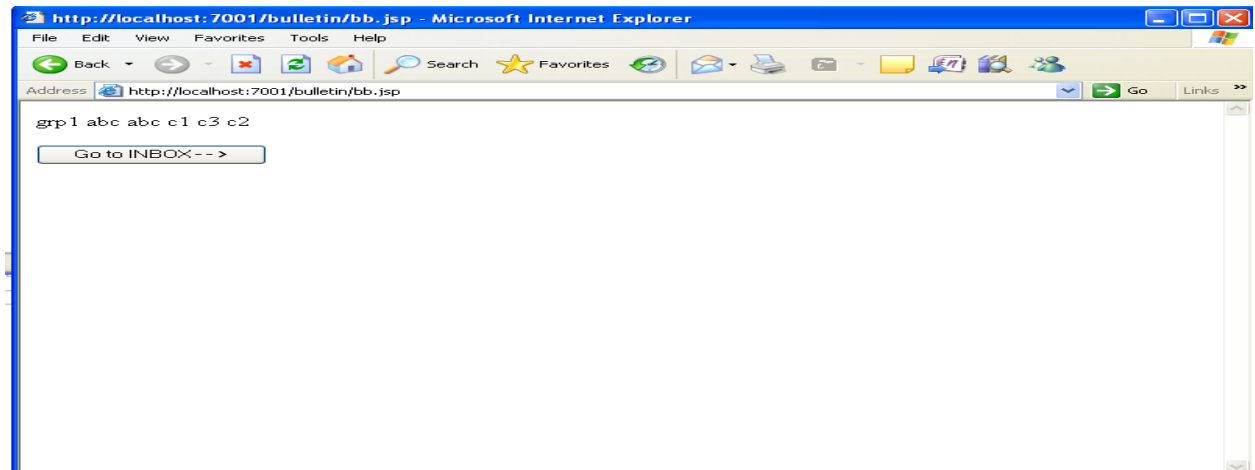
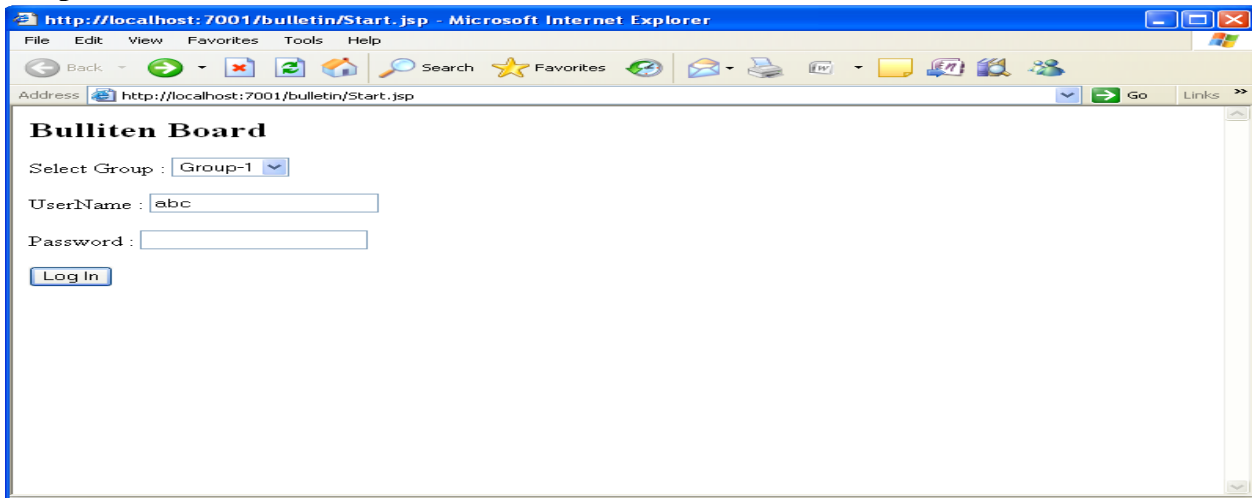
### **Update.jsp:**

```
<%@ page import="java.sql.*" %>  
<%  
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
String flag=request.getParameter("flag");  
String msg=request.getParameter("msg");  
try  
{  
Connection con  
=DriverManager.getConnection("jdbc:odbc:myserver","system","manager");  
PreparedStatement stmt=con.prepareStatement("insert into "+flag+" values '"+msg+"'");  
int rs=stmt.executeUpdate();  
if(rs == 1)  
out.println("written successfully");  
%>  
<br><form method="post" action="Inbox.jsp" >  
<input type="hidden" name="flag" value="<%=flag%>" />  
<input type="button" value="< - - Go back to INBOX" onclick="submit()" />  
</form>  
<%  
stmt.close();  
}  
catch(SQLException se)  
{  
%>  
Database ERROR  
<%  
}  
%>
```

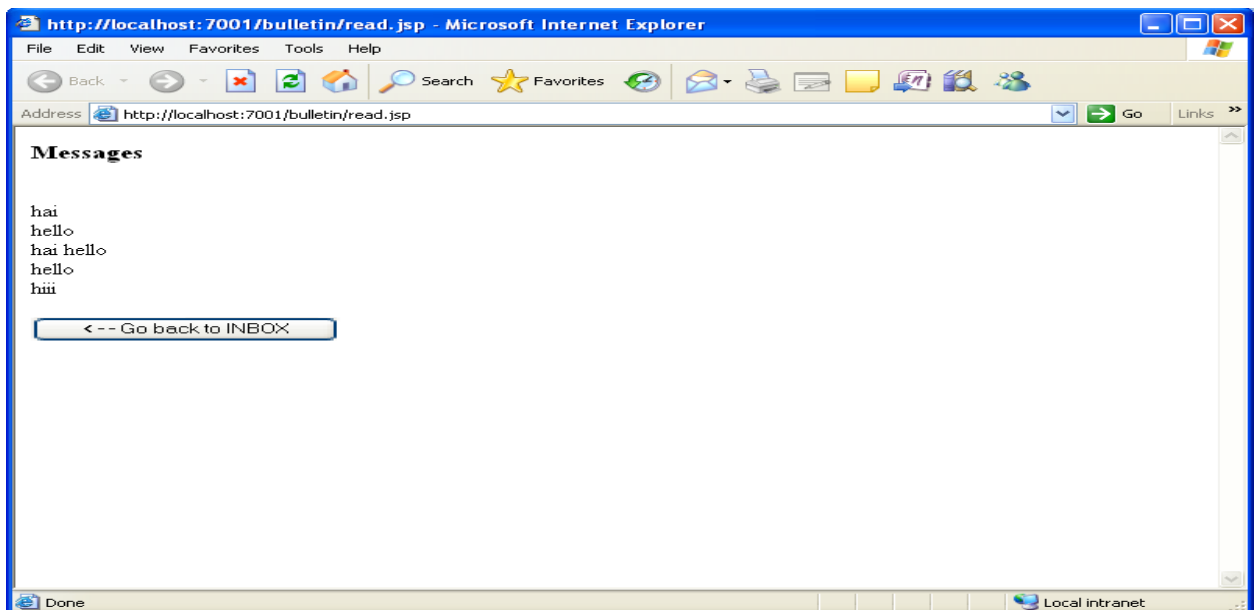
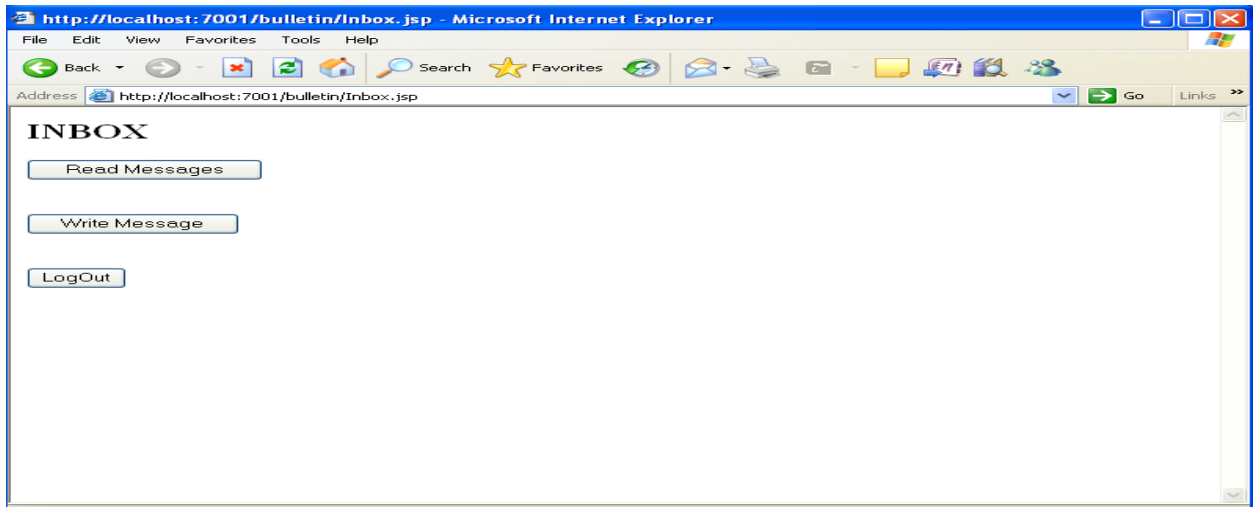
## Tables required –

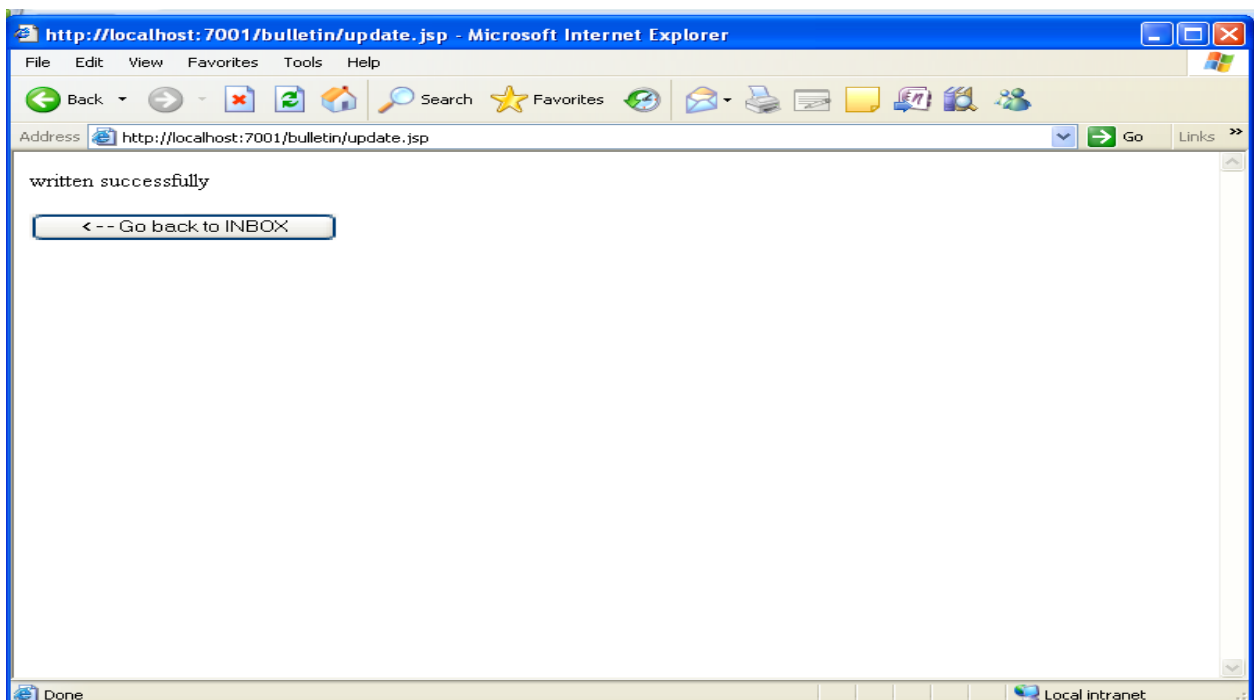
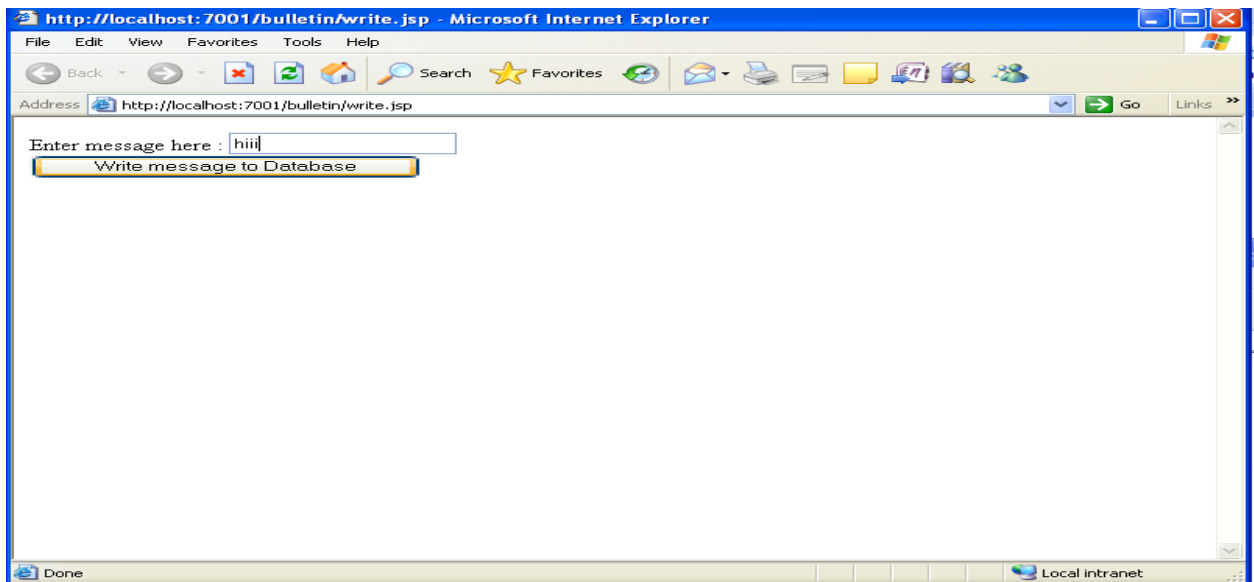
1. Login (grp varchar2(20), usr varchar2(20), pass varchar2(20));
2. grp1 (msg varchar2(50));
3. grp2 (msg varchar2(50));

## Output:









## 9. Implement client-server communication using UDP

### Introduction:

With UDP, computer applications can send messages, in this case referred to as *datagrams*, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. UDP uses a simple transmission model without implicit hand-shaking for providing reliability, ordering, or data integrity. UDP's stateless nature is also useful for servers answering small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).

Common network applications that use UDP include the Domain Name System (DNS).

UDP applications use **datagram sockets** to establish host-to-host communications. A **Datagram socket** is a type of Internet socket, which is the sending or receiving point for packet delivery services.[1] Each packet sent or received on a Datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may arrive in any order and might not arrive at the receiving computer. UDP broadcasts sends are always enabled on a Datagram Socket. In order to receive broadcast packets a Datagram Socket must be bound to a more specific address.

An application binds a socket to its endpoint of data transmission, which is a combination of an IP address and a service port. A port is a software structure that is identified by the port number, a 16 bit integer value, allowing for port numbers between 0 and 65535.

Port numbers are divided into three ranges:

- Port numbers 0 through 1023 are used for common, well-known services.
- Port numbers 1024 through 49151 are the registered ports
- Ports 49152 through 65535 are dynamic ports that are not officially used for any specific service, and can be used for any purpose. They are also used as ephemeral ports, from which software running on the host may randomly choose a port in order to define itself.[2] In effect, they are used as temporary ports primarily by clients when communicating with servers.

### **UDP Server:**

```
import java.net.*;
import java.io.*;
public class UDPServer {
    public static void main(String args[]) {
        DatagramSocket aSocket = null;
        try {
            aSocket = new DatagramSocket(8117);
            byte[] buffer = new byte[1000];
            while(true)
            {
                DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(request);
                DatagramPacket reply = new DatagramPacket(request.getData(),
                    request.getLength(),
                    request.getAddress(), request.getPort());
                aSocket.send(reply);
                System.out.println("Reply:" + new String(reply.getData()));
            }
        }
        catch (SocketException e)
        {
            System.out.println("Socket: " + e.getMessage());
        }
        catch (IOException e)
        {
            System.out.println("IO: " + e.getMessage());
        }
        finally
        {
            if(aSocket != null) aSocket.close();
        }
    }
}
```

### **UDP Client:**

```
import java.net.*;
import java.io.*;
public class UDPClient {
    public static void main(String args[]) {
        DatagramSocket aSocket=null;
        try {
            aSocket=new DatagramSocket();
            byte[] m=args[0].getBytes();
            InetAddress aHost=InetAddress.getByName(args[1]);
            int serverPort=8117;
            DatagramPacket request=new DatagramPacket(m,args[0].length(),aHost,serverPort);
```

```

aSocket.send(request);
byte[] buffer=new byte[1000];
DatagramPacket reply=new DatagramPacket(buffer,buffer.length);
aSocket.receive(reply);
System.out.println("Reply:"+new String(reply.getData()));
}
catch(SocketException e)
{ System.out.println("Socket:"+e.getMessage()); }
catch(IOException e)
{ System.out.println("IO:"+e.getMessage()); }
finally
{
if(aSocket!=null)
aSocket.close();
}
}
}

```

## UDP: OUTPUT

C:\WINDOWS\system32\cmd.exe - java UDPServer 127.0.0.1

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

C:\Documents and Settings\admin>d:

D:\>cd udp

D:\udp>javac \*.java

D:\udp>java UDPServer 127.0.0.1  
Reply:hello

C:\WINDOWS\system32\cmd.exe

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

C:\Documents and Settings\admin>d:

D:\>cd udp

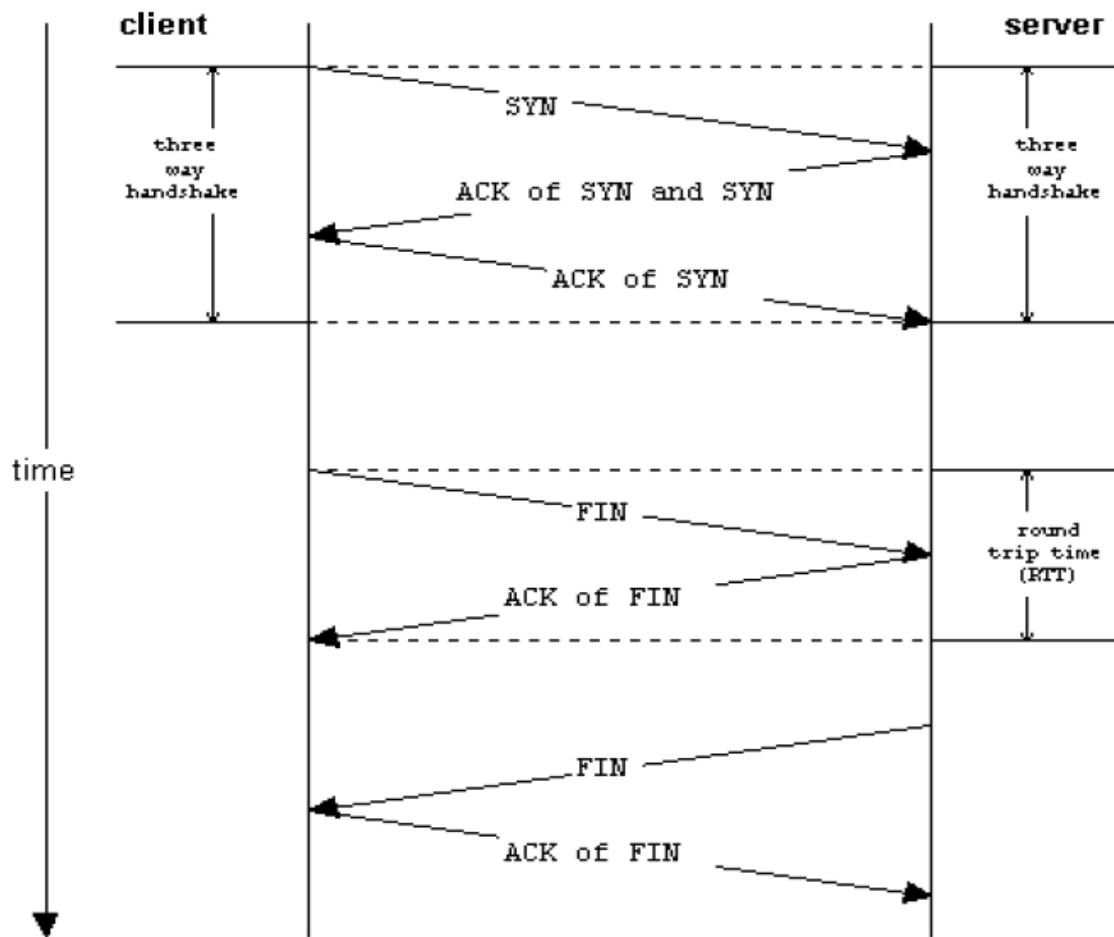
D:\udp>java UDPClient hello 127.0.0.1  
Reply:hello

## 10. Implement client-server communication using TCP

### Introduction:

TCP provides the service of exchanging data directly between two network hosts, whereas IP handles addressing and routing message across one or more networks. In particular, TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer.

TCP is the protocol that major Internet applications rely on, applications such as the World Wide Web, e-mail, and file transfer. TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the *Internet socket*. During the lifetime of a TCP connection it undergoes a series of state changes:



1. LISTEN : In case of a server, waiting for a connection request from any remote client.
2. SYN-SENT : waiting for the remote peer to send back a TCP segment with the SYN and ACK flags set. (usually set by TCP clients)
3. SYN-RECEIVED : waiting for the remote peer to send back an acknowledgment after having sent back a connection acknowledgment to the remote peer. (usually set by TCP servers)
4. ESTABLISHED : the port is ready to receive/send data from/to the remote peer.
5. FIN-WAIT-1
6. FIN-WAIT-2
7. CLOSE-WAIT
8. CLOSING
9. LAST-ACK
10. TIME-WAIT : represents waiting for enough time to pass to be sure the remote peer received the acknowledgment of its connection termination request. According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes.
11. CLOSED

TCP uses the notion of port numbers to identify sending and receiving application end-points on a host, or *Internet sockets*. Each side of a TCP connection has an associated 16-bit unsigned port number (0-65535) reserved by the sending or receiving application. Arriving TCP data packets are identified as belonging to a specific TCP connection by its sockets, that is, **the combination of source host address, source port, destination host address, and destination port**. This means that a server computer can provide several clients with several services simultaneously, as long as a client takes care of initiating any simultaneous connections to one destination port from different source ports.

### TCP Server:

```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main(String a[])
    {
        Socket s=null;
        try
        {
            int serverPort=8117;
            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);
        System.out.println("Received: " + 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("IO:"+e.getMessage()); }
    }
}
}

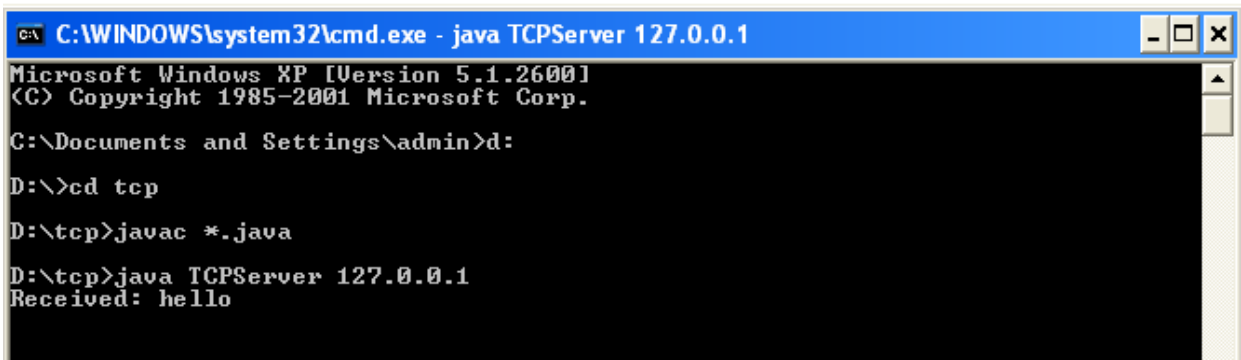
```



## TCP Client:

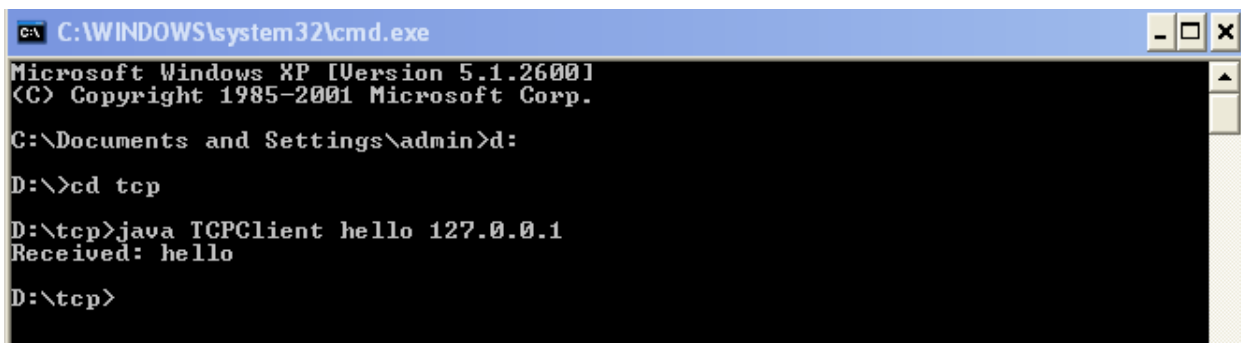
```
import java.net.*;
import java.io.*;
public class TCPClient {
    public static void main (String args[]) {
        // arguments supply message and hostname
        Socket s = null;
        try{
            int serverPort = 8117;
            s = new Socket(args[1],serverPort);
            DataInputStream in = new DataInputStream( s.getInputStream());
            DataOutputStream out =new DataOutputStream( s.getOutputStream());
            out.writeUTF(args[0]); // UTF is a string encoding see Sn. 4.4
            String data = in.readUTF(); // read a line of data from the stream
            System.out.println("Received: "+ 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("readline:"+e.getMessage());}
        }finally {if(s!=null) try {s.close();}catch (IOException
        e){System.out.println("close:"+e.getMessage());}}
    }
}
```

## TCP: OUTPUT



```
C:\WINDOWS\system32\cmd.exe - java TCPServer 127.0.0.1
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\admin>d:
D:\>cd tcp
D:\tcp>javac *.java
D:\tcp>java TCPServer 127.0.0.1
Received: hello
```



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

C:\Documents and Settings\admin>d:
D:\>cd tcp
D:\tcp>java TCPClient hello 127.0.0.1
Received: hello
D:\tcp>
```

# **ADDITIONAL PROGRAMS**

## **1) AIM : To implement Remote Method Invocation (RMI)**

**DESCRIPTION :** RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a *distributed object application*.

Distributed object applications need to do the following:

Locate remote objects. Applications can use various mechanisms to obtain references to remote objects. For example, an application can register its remote objects with RMI's simple naming facility, the RMI registry. Alternatively, an application can pass and return remote object references as part of other remote invocations.

Communicate with remote objects. Details of communication between remote objects are handled by RMI. To the programmer, remote communication looks similar to regular Java method invocations.

Load class definitions for objects that are passed around. Because RMI enables objects to be passed back and forth, it provides mechanisms for loading an object's class definitions as well as for transmitting an object's data.

## **WRITE RMI PROGRAM FOR ARITHMETIC OPERATIONS PROGRAM :**

**// Calculator.java for interface**

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface Calculator extends Remote  
{  
    public long addition(long a,long b) throws RemoteException; public  
    long subtraction(long a,long b) throws RemoteException; public  
    long multiplication(long a,long b) throws RemoteException;
```

```

    public long division(long a,long b) throws RemoteException;
}

```

#### **// Calculatorimpl.java**

```

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class CalculatorImpl extends UnicastRemoteObject implements Calculator
{
    protected CalculatorImpl() throws RemoteException
    {
        super();
    }
    public long addition(long a, long b) throws RemoteException
    {
        return a+b;
    }
    public long subtraction(long a, long b) throws RemoteException
    {
        return a-b;
    }
    public long multiplication(long a, long b) throws RemoteException
    {
        return a*b;
    }
    public long division(long a, long b) throws RemoteException
    {
        return a/b;
    }
}

```

#### **// Calculatorclient.java import**

```

java.rmi.Naming;
public class CalculatorClient
{
    public static void main(String[] args)
    {
        try
        {
            Calculator c = (Calculator)
Naming.lookup("//127.0.0.1:1099/CalculatorService");
            System.out.println("Addition : " +c.addition(10,5)); System.out.println("Subtraction : "
+c.subtraction(10,5)); System.out.println("Multiplication : " +c.multiplication(10,5));
            System.out.println("Division : " +c. division(10,5));
        }
        catch (Exception e)

```

```

        {
            System.out.println("Exception is : "+e);
        }
    }
}

// CalculatorServer.java
import java.rmi.Naming; public class CalculatorServer
{
    CalculatorServer()
    {
        try
        {
            Calculator c = new CalculatorImpl();
            Naming.rebind("rmi://localhost:1099/CalculatorService", c);
        }
        catch (Exception e)
        {
            System.out.println("Exception is : "+e);
        }
    }
    public static void main(String[] args)
    {
        new CalculatorServer();
    }
}

```

### **Execution Procedure**

Compile all programs using javac

Generate the stub file which is used as proxy for client using rmic command Next start the rmiregistry

Start the server and next start the client

```

C:\murari>javac Calculator.java
C:\murari>javac CalculatorImpl.java
C:\murari>javac CalculatorServer.java
C:\murari>
C:\murari>javac CalculatorClient.java
C:\murari>java CalculatorClient
Addition : 15
Subtraction : 5
Multiplication :50
Division : 2
C:\murari>

C:\murari>rmic CalculatorImpl
C:\murari>start rmiregistry
C:\murari>java CalculatorServer
C:\murari>java CalculatorServer

```

## 2) AIM : RMI program to implement Domain Name System (DNS)

**DESCRIPTION:** The Domain Name System (DNS) is a hierarchical distributed naming system 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. A Domain Name Service resolves queries for these names into IP addresses for the purpose of locating computer services and devices worldwide. By providing a worldwide, distributed keyword-based redirection service, the Domain Name System is an essential component of the functionality of the Internet.

**AIM : To write a RMI program for implementing DNS.**

**PROGRAM :**

**// CLIENT PROGRAM**

```
import java.rmi.*; public class
AddClient {
    public static void main(String args[]) { try {
        String addServerURL = "rmi://" + args[0] + "/AddServer";
        AddServerIntf addServerIntf = (AddServerIntf)Naming.lookup(addServerURL);
        System.out.println("The dns is: " + addServerIntf.getDns());
    }
    catch(Exception e) {
        System.out.println("Exception: " + e);
    }
}
}
```

**//SERVER PROGRAM** import

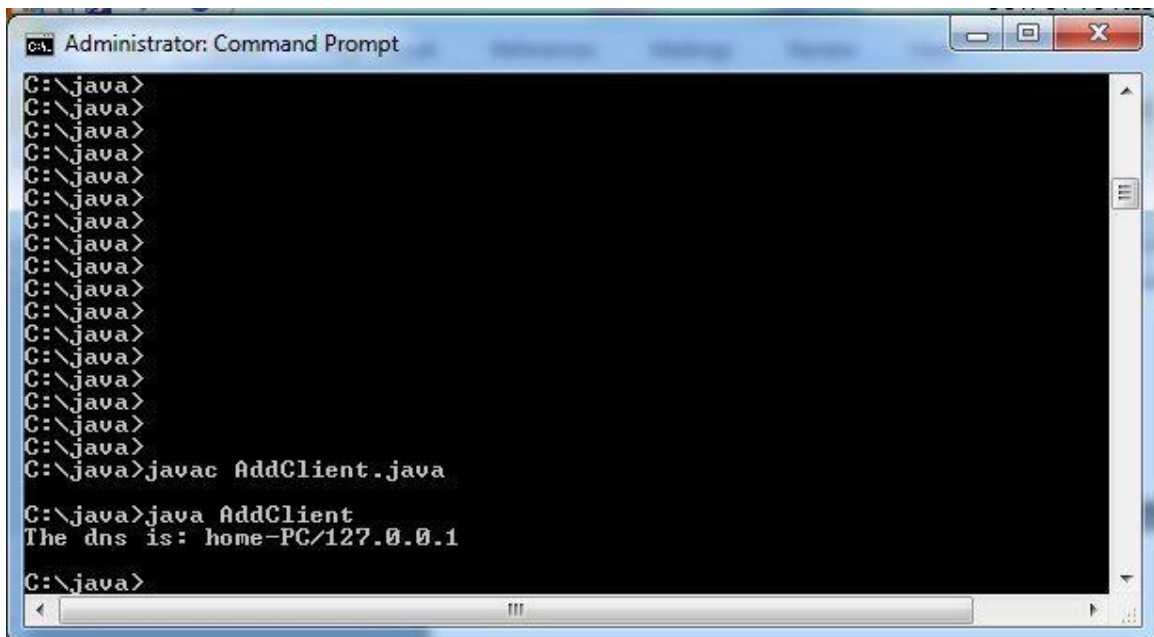
```
java.net.*; import java.rmi.*;
public class AddServer {
    public static void main(String args[]) { try {
        AddServerImpl addServerImpl = new AddServerImpl();
        Naming.rebind("AddServer", addServerImpl);
    }
    catch(Exception e) {
        System.out.println("Exception: " + e);
    }
}
}
```

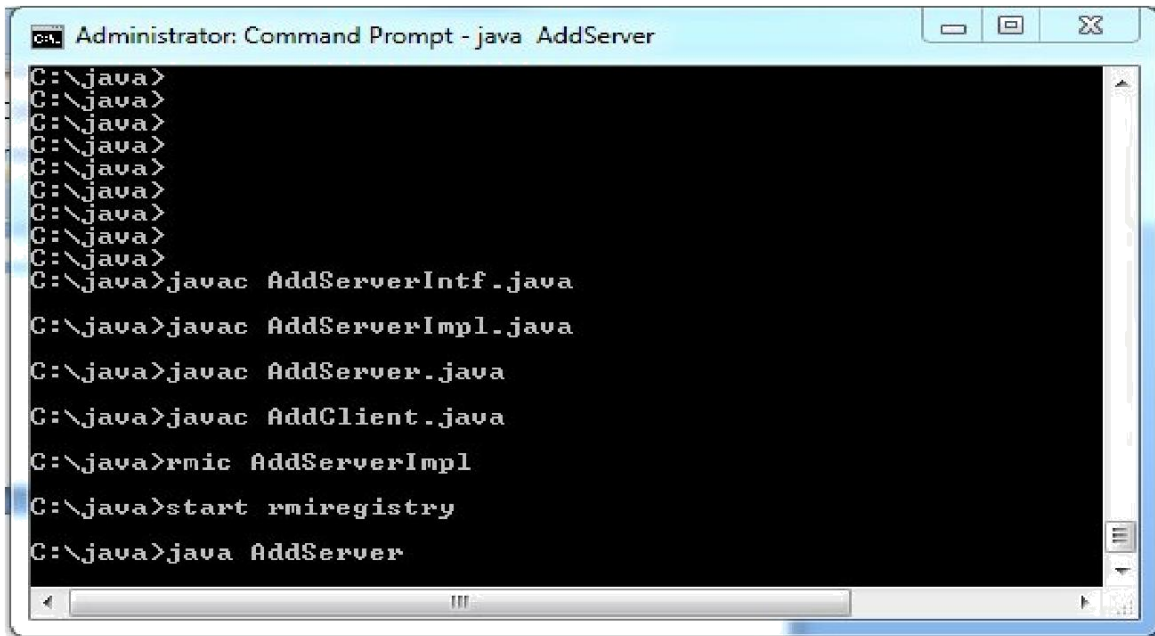
```
import java.rmi.*;
import java.net.*; import
java.rmi.server.*;
public class AddServerImpl extends UnicastRemoteObject implements AddServerIntf {
    InetAddress[] address;

    public AddServerImpl() throws RemoteException {
    }

    public String getDns() throws RemoteException { try
    {
        address=InetAddress.getAllByName("localhost");
    }
    catch(Exception e) {
        System.out.println("Exception: " + e);
    }
    return address.toString();
    }
}
```

## OUTPUT : DNS USING RMI





```
Administrator: Command Prompt - java AddServer
C:\java>
C:\java>
C:\java>
C:\java>
C:\java>
C:\java>
C:\java>
C:\java>
C:\java>
C:\java>javac AddServerIntf.java
C:\java>javac AddServerImpl.java
C:\java>javac AddServer.java
C:\java>javac AddClient.java
C:\java>rmic AddServerImpl
C:\java>start rmiregistry
C:\java>java AddServer
```

### Steps for implementing in WebLogic platform

- 1) Go to WebLogic Platform in programs/All programs, then select *'Configuration Wizard'*.
- 2) Select *'create a new WebLogic Configuration'* then click *next* button.
- 3) Then a list of domains appear where *Weblogic Server Domain* will be selected. Don't change it. Click *next* button.
- 4) In the next page, Select *Express* then click *'next'*.
- 5) Give a username, password atleast 8 characters then click *'next'*.
- 6) Give a configuration name then click *'create'*, this name will be used to start the server.
- 7) Click *done* or *exit* after completing all steps.

### Starting the server

- 1) Go to *WeblogicPlatform* select *'userprojects'* then *start server*, here we can see the given configuration name at step 6.
- 2) A command prompt appears, wait till it appears in *RUNNING MODE*.

### Running the Program

- 1) Go to *WeblogicPlatform* select *WebLogicBuilder*.
- 2) From toolbar menu click *Tools*, then connect to server.
- 3) Enter Username, password given at step5, click '*connect*'.
- 4) Click '*File*' and click '*open*', browse the location of jsp files.

If descriptors are available then the above jsp location will appear in blue colour.  
Otherwise, it will ask for creating them.

- 5) From menu, click *Tools* , then click '*Deploy module*'. A dialog box appears again click on 'Deploy module'.
- 6) Check the status till it deploys successfully.
- 7) Open a browser, type the URL, execute.....