Name: Abhay Magar

Roll no.: 34

Batch: B2

Experiment No. 1

Title: Implementation of Client/Server using Socket.

Code:

Server.java

```
import java.util.*;
import java.io.*;
import java.net.*;
public class Server {
    public static void main(String args[]) throws Exception{
            //Server server = new Server();
            ServerSocket MyServer = new ServerSocket(25);
            Socket ss = MyServer.accept();
            DataInputStream din =new DataInputStream(ss.getInputStream());
            DataOutputStream dout=new DataOutputStream(ss.getOutputStream());
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        Server server = new Server();
        String str="",str2="";
        int sum = 0;
            while(!str.equals("stop")){
                    str=din.readUTF();
                    if(str.equals("stop"))
                            break;
                    sum = sum + Integer.parseInt(str);
            }
            dout.writeUTF(Integer.toString(sum));
            dout.flush();
            din.close();
            ss.close();
            MyServer.close();
```

```
    }
}
```

Client.java

```
import java.io.*;
import java.util.*;
import java.net.*;
public class Client
{
        public static void main(String args[])throws Exception
        {
                String send="",r="";
                Socket MyClient = new Socket("192.168.0.106",25);
                DataInputStream din=new DataInputStream(MyClient.getInputStream());
                DataOutputStream dout = new
DataOutputStream(MyClient.getOutputStream());
                Scanner sc = new Scanner(System.in);
                while(!send.equals("stop")){
                        System.out.print("Send: ");
                        send = sc.nextLine();
                        dout.writeUTF(send);
                }
                dout.flush();
                r=din.readUTF();
                System.out.println("Reply: "+ r);
                dout.close();
                din.close();
                MyClient.close();
        }
}
```

Output:

C:\Windows\System32\cmd.exe - java Server

```
D:\Codes\Java\Sem 8\Experiment 1>javac Server.java

D:\Codes\Java\Sem 8\Experiment 1>java Server
```

C:\Windows\System32\cmd.exe - java Client

```
D:\Codes\Java\Sem 8\Experiment 1>javac Client.java

D:\Codes\Java\Sem 8\Experiment 1>java Client
Send:
```

C:\Windows\System32\cmd.exe

```
D:\Codes\Java\Sem 8\Experiment 1>javac Server.java

D:\Codes\Java\Sem 8\Experiment 1>java Server

D:\Codes\Java\Sem 8\Experiment 1>
```

C:\Windows\System32\cmd.exe

```
D:\Codes\Java\Sem 8\Experiment 1>javac Client.java

D:\Codes\Java\Sem 8\Experiment 1>java Client
Send: 1
Send: 2
Send: 3
Send: 4
Send: 5
Send: stop
Reply: 15

D:\Codes\Java\Sem 8\Experiment 1>
```

Experiment No. 2

Title: Implementation of Client/Server using RPC/RMI.

Interface

```
import java.rmi.*;
public interface SmallI extends Remote{
	public int small(int x,int y)throws RemoteException;
}
```

Interface Implementation

```
import java.rmi.*;
import java.rmi.server.*;
public class Small extends UnicastRemoteObject implements SmallI
{
	Small() throws RemoteException{
		super();
	}
	public int small(int x,int y){
		if(x<y)
			return x;
		else
			return y;
	}
}
```

Server.java

```
import java.rmi.*;
import java.rmi.registry.*;
public class Server{
	public static void main(String args[]){
		try{
			SmallI stub=new Small();
			Naming.rebind("rmi://localhost:/small",stub);
		}
		catch(Exception e){
			System.out.println(e);
		}
	}
}
```

Client

```
import java.rmi.*;
```

```
import java.util.*;
public class Client{
        public static void main(String args[]){
                try{
                        int n1=0,n2=0;
                        Scanner sc=new Scanner(System.in);
                        SmallI stub=(SmallI)Naming.lookup("rmi://localhost:/small");
                        System.out.println("Enter the Number 1");
                        n1=sc.nextInt();
                        System.out.println("Enter the Number 2");
                        n2=sc.nextInt();
                        System.out.println("Smallest Number is:-"+(stub.small(n1,n2)));
                }
                catch(Exception e){
                        System.out.println("Error is "+e);
                }
        }
}
```

Output

```
C:\Windows\system32\cmd.exe
                        Adder stub=new AdderRemote();
                                       ^
  symbol:   class AdderRemote
  location: class Server
2 errors

C:\Users\sakec\Desktop\BE-3-14>javac *.java

C:\Users\sakec\Desktop\BE-3-14>rmic Small
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

C:\Users\sakec\Desktop\BE-3-14>start rmiregistry

C:\Users\sakec\Desktop\BE-3-14>java Client
Enter the Number 1
45
Enter the Number 2
50
Smallest Number is:-45

C:\Users\sakec\Desktop\BE-3-14>
```

```
C:\Windows\system32\cmd.exe - java Server

C:\Users\sakec\Desktop\BE-3-14>java Server
```

Experiment No. 3

Title: Implementation of Bully Election Algorithm.

Code:

```
import java.io.*;

import java.util.Scanner;


class Main{

    static int n;

    static int pro[] = new int[100];

    static int sta[] = new int[100];

    static int co;


    public static void main(String args[])throws IOException

    {

        System.out.println("Enter the number of process");

        Scanner in = new Scanner(System.in);

        n = in.nextInt();


        int i,j,k,l,m;


        for(i=0;i<n;i++)

        {

            System.out.println("For process "+(i+1)+":");

            System.out.println("Status:");

            sta[i]=in.nextInt();

            System.out.println("Priority");

            pro[i] = in.nextInt();

        }
```

```
        System.out.println("Which process will initiate election?");
        int ele = in.nextInt();
        elect(ele);
        System.out.println("Final coordinator is "+co);
    }

    static void elect(int ele)
    {
        ele = ele-1;
        co = ele+1;
        for(int i=0;i<n;i++)
        {
            if(pro[ele]<pro[i])
            {
                System.out.println("Election message is sent from "+(ele+1)+" to "+(i+1));
                if(sta[i]==1)
                    elect(i+1);
            }
        }
    }
}
```

Output:

```
Enter the number of process
6
For process 1:
Status:
1
Priority
4
For process 2:
Status:
5
Priority
3
For process 3:
Status:
7
Priority
2
For process 4:
Status:
6
Priority
3
For process 5:
Status:
1
Priority
1
For process 6:
Status:
9
Priority
2
Which process will initiate election?
6
Election message is sent from 6 to 1
Election message is sent from 6 to 2
Election message is sent from 6 to 4
Final coordinator is 1
```

Experiment No. 4

Title: Implementation of Clock Synchronization using Berkley Algorithm.

Code:

```
import java.io.*;
import java.util.*;
public class Berkeley {
        float diff(int h, int m, int s, int nh, int nm, int ns) {
        int dh = h - nh;
        int dm = m - nm;
        int ds = s - ns;
        int diff = (dh * 60 * 60) + (dm * 60) + ds;
        return diff;
    }
    float average(float diff[], int n) {
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += diff[i];
        }
        float average = (float) sum / (n + 1);
        System.out.println("The average of all time differences is " + average);
        return average;
    }
    void sync(float diff[], int n, int h, int m, int s, int nh[], int nm[], int ns[], float average) {
        for (int i = 0; i < n; i++) {
            diff[i] += average;
            int dh = (int) diff[i] / (60 * 60);
```

```
diff[i] %= (60 * 60);
int dm = (int) diff[i] / 60;
diff[i] %= 60;
int ds = (int) diff[i];
nh[i] += dh;
if (nh[i] > 23) {
        nh[i] %= 24;
}
nm[i] += dm;
if (nm[i] > 59) {
        nh[i]++;
        nm[i] %= 60;
}
ns[i] += ds;
if (ns[i] > 59) {
        nm[i]++;
        ns[i] %= 60;
}
if (ns[i] < 0) {
        nm[i]–;
        ns[i] += 60;
}
}
h += (int)(average / (60 * 60));
if (h > 23) {
        h %= 24;
}
m += (int)(average / (60 * 60 * 60));
if (m > 59) {
```

```
            h++;
            m%=60;
        }
        s+=(int)(average%(60*60*60));
        if(s>59){
            m++;
            s%=60;
        }
        if(s<0){
            m–;
            s+=60;
        }
        System.out.println("The synchronized clocks are:\nTime Server–>"+h+":"+m+":"+s);
        for(int i=0;i<n;i++){
        System.out.println("Node"+(i+1)+"–>"+nh[i]+":"+nm[i]+":"+ns[i]);
            }
    }
    public static void main(String[] args) throws IOException {
        Berkeley b=new Berkeley();
        Date date=new Date();
        BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter number of nodes:");
        int n=Integer.parseInt(obj.readLine());
        int h=date.getHours();
        int m=date.getMinutes();
        int s=date.getSeconds();
        int nh[]=new int[n];
        int nm[]=new int[n];
        int ns[]=new int[n];
```

```
for (int i = 0; i < n; i++) {
        System.out.println("Enter time for node " + (i + 1) + "\n Hours:");
        nh[i] = Integer.parseInt(obj.readLine());
        System.out.println("Minutes:");
        nm[i] = Integer.parseInt(obj.readLine());
        System.out.println("Seconds:");
        ns[i] = Integer.parseInt(obj.readLine());
    }
    for (int i = 0; i < n; i++) {
        System.out.println("Time Server sent time " + h + ":" + m + ":" + s + " to node " + (i +
        1));
    }
    float diff[] = new float[n];
    for (int i = 0; i < n; i++) {
        diff[i] = b.diff(h, m, s, nh[i], nm[i], ns[i]);
        System.out.println("Node " + (i + 1) + " sent time difference of " + (int) diff[i] + " to
        Time Server.");
    }
    float average = b.average(diff, n);
    b.sync(diff, n, h, m, s, nh, nm, ns, average);
    }
}
```

Output:

```
Dell@DESKTOP-F1A1TMK MINGW64 /d/Notes/Colle
ge/SEM 8/Practical/DC/EXP4
$ java Berkeley
Enter number of nodes:
2
Enter time for node 1
 Hours:
11
Minutes:
48
Seconds:
45
Enter time for node 2
 Hours:
11
Minutes:
49
Seconds:
41
Time Server sent time 18 : 37 : 22 to node 1
Time Server sent time 18 : 37 : 22 to node 2
Node 1 sent time difference of 24517 to Time Server.
Node 2 sent time difference of 24461 to Time Server.
The average of all time differences is 16326.0
The synchronized clocks are:
Time Server ---> 22 : 38 : 28
Node 1 ---> 23 : 9 : 28
Node 2 ---> 23 : 9 : 28
```

Experiment No. 5

Title: Implementation of Lamport's algorithm Mutual Exclusion.

Code:

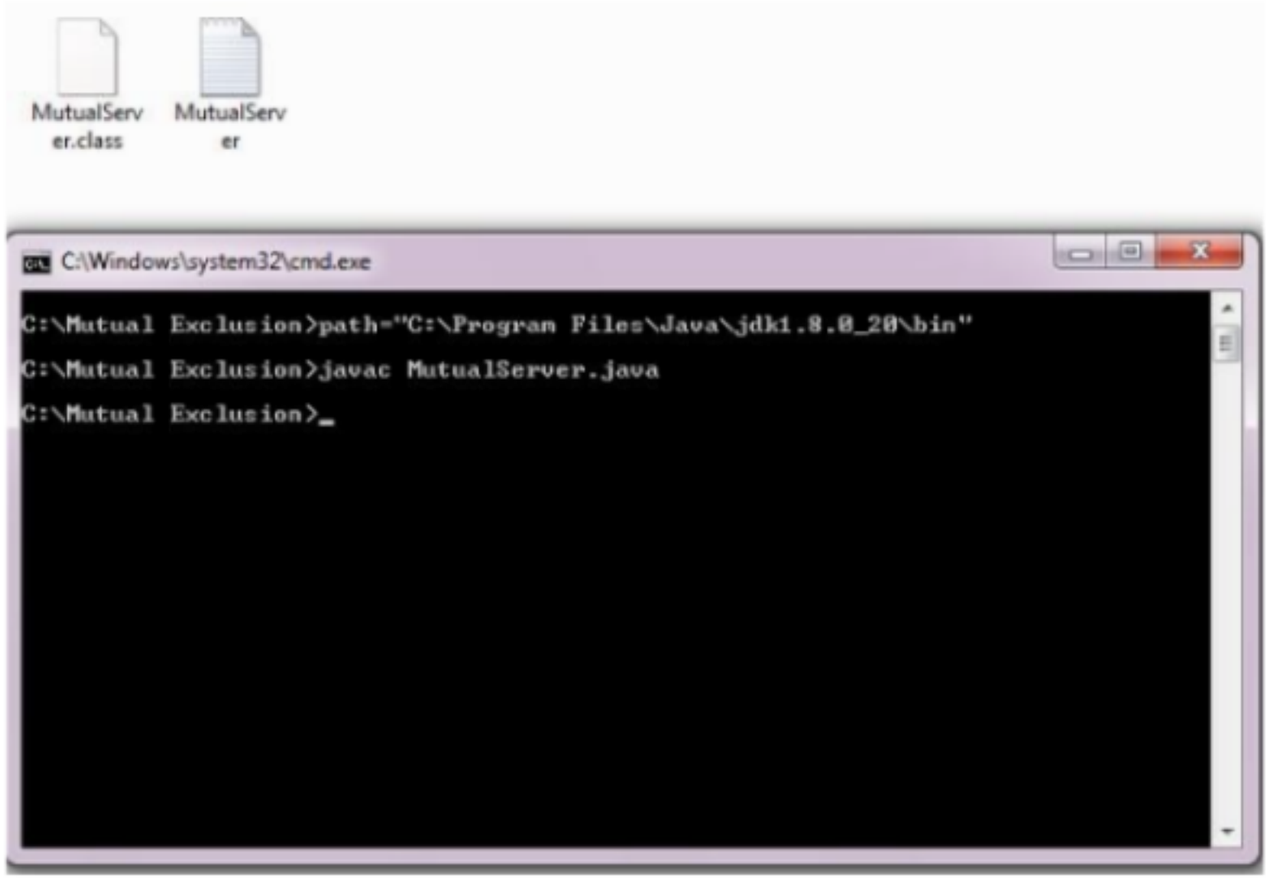
Step 1 – Write and Compile Server program MutualServer.java.

MutualServer.java.

```
import java.io.*;

import java.net.*;

public class MutualServer implements Runnable {

Socket socket=null;

static ServerSocket ss;

MutualServer(Socket newSocket){

this.socket=newSocket;
```

```
}
public static void main(String args[]) throws IOException
{
ss=new ServerSocket(7000);
System.out.println("Server Started");
while(true){
Socket s = ss.accept();
MutualServer es = new MutualServer(s);
Thread t = new Thread(es);
t.start();
}
}
public void run(){
try {
BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
while(true){
System.out.println(in.readLine());
}
}catch(Exception e){}
}
}
```

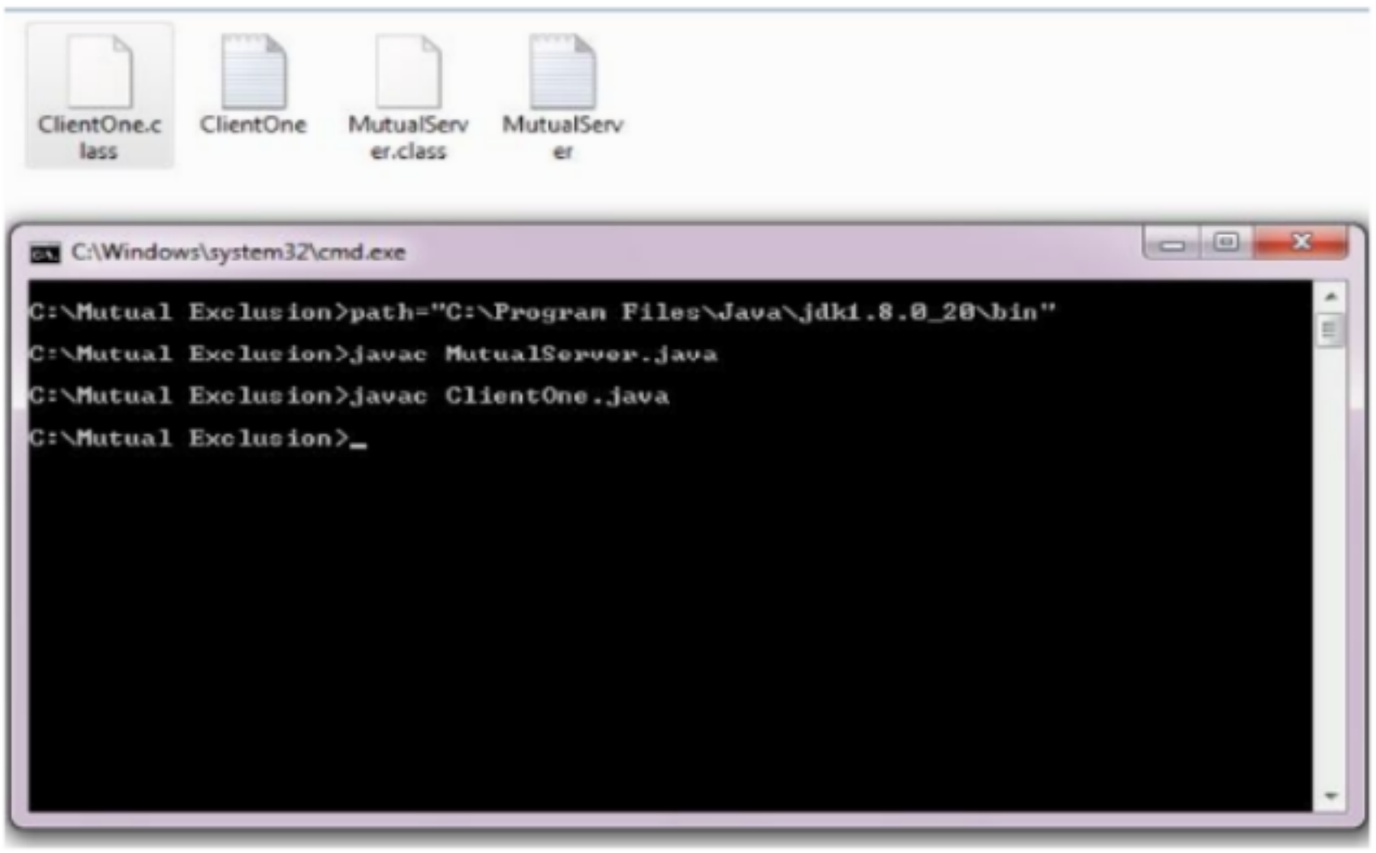
Compile Server Program as follows:



Step 2 – Write and Compile First client program ClientOne.java. ClientOne.java

```
import java.io.*;
import java.net.*;
public class ClientOne{
public static void main(String args[])throws IOException{
Socket s=new Socket("localhost",7000);
PrintStream out = new PrintStream(s.getOutputStream());
ServerSocket ss = new ServerSocket(7001);
Socket s1 = ss.accept();
BufferedReader in1 = new BufferedReader(new InputStreamReader(s1.getInputStream()));
PrintStream out1 = new PrintStream(s1.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str="Token";
while(true){
if(str.equalsIgnoreCase("Token")){
System.out.println("Do you want to send some data");
System.out.println("Enter Yes or No");
str=br.readLine();
if(str.equalsIgnoreCase("Yes")){
```

```
System.out.println("Enter the Data");
str=br.readLine();
System.out.println(str);
}
System.out.println("Token");
}
System.out.println("Waiting for Token");
str=in1.readLine();
}
}
}
```

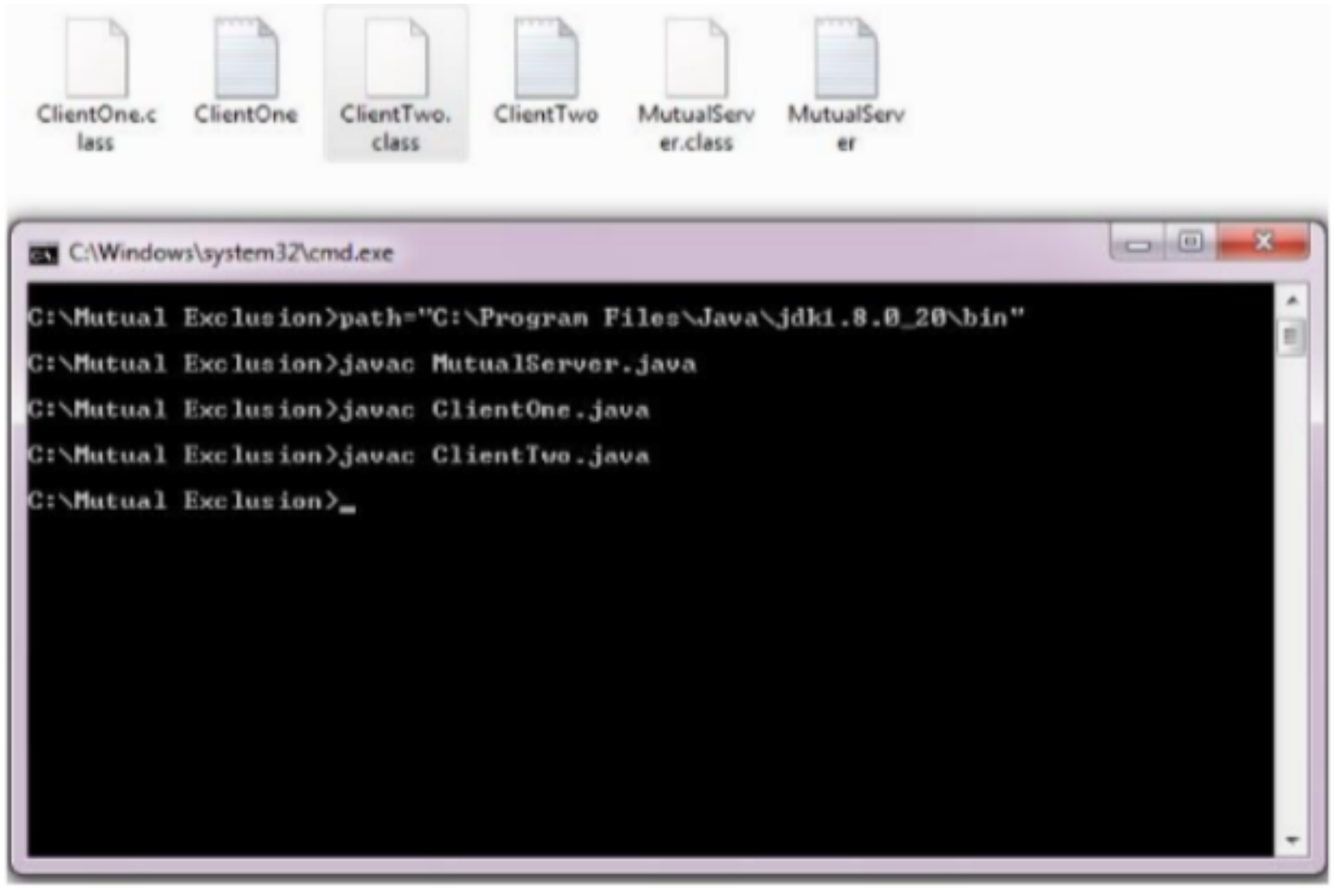


Step 3 – Write and Compile Second client program ClientTwo.java.

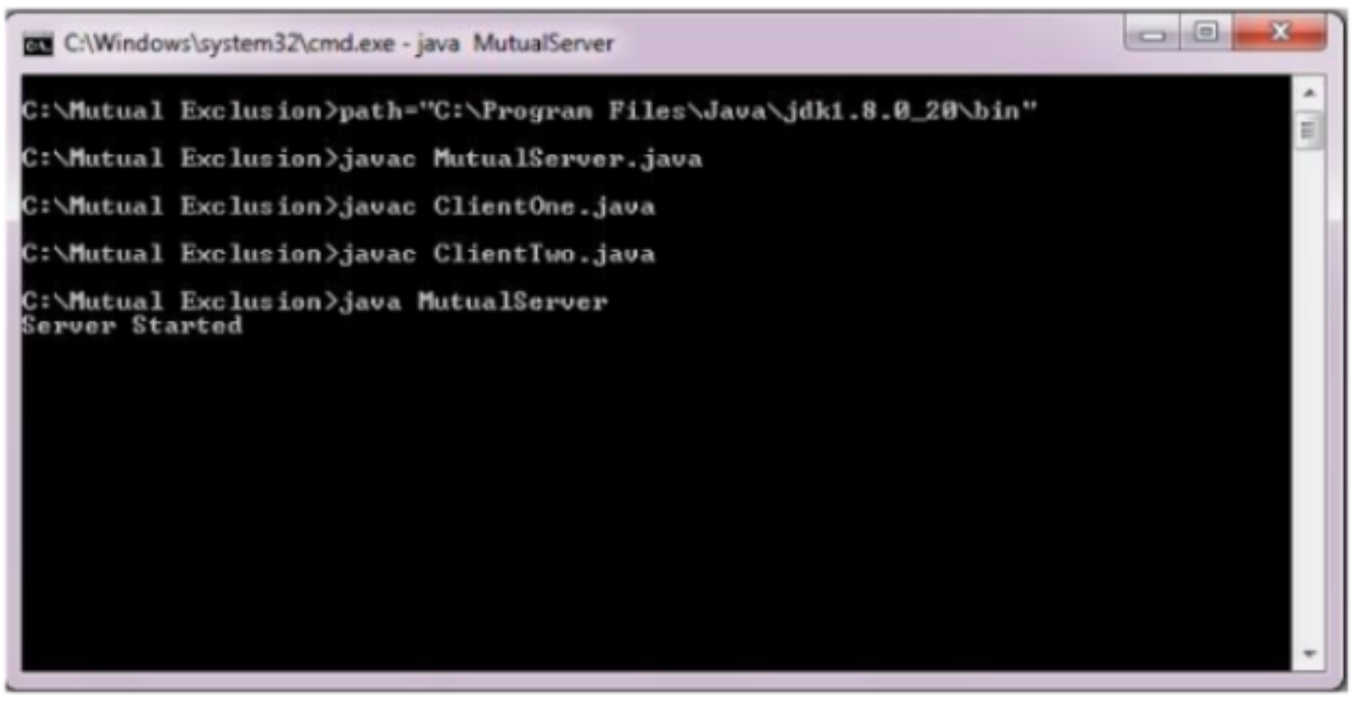
ClientTwo.java

```
import java.io.*;
import java.net.*;
public class ClientTwo{
public static void main(String args[])throws IOException{
Socket s = new Socket("localhost",7000);
PrintStream out = new PrintStream(s.getOutputStream());
```

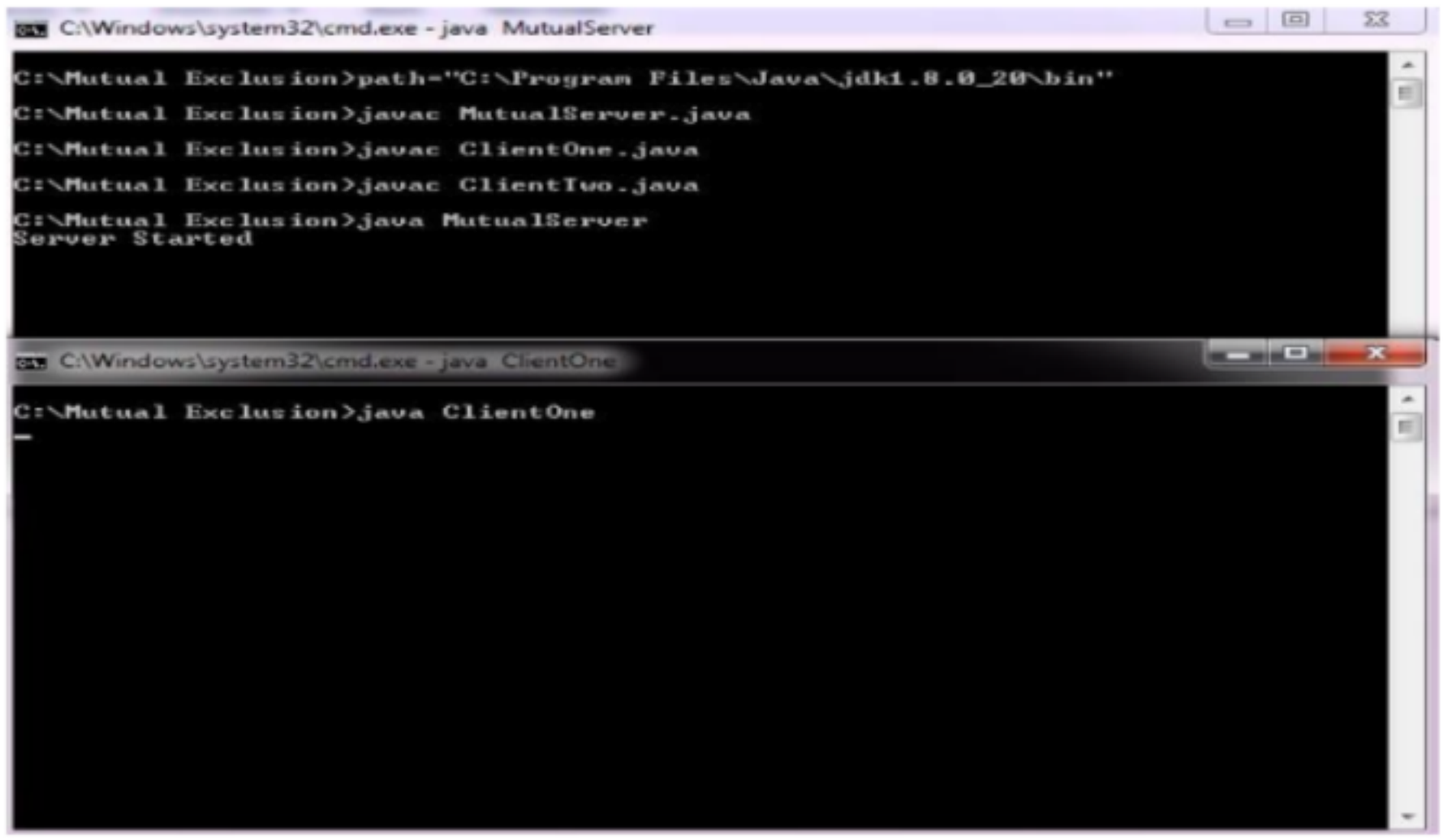
```
Socket s2=new Socket("localhost",7001);
BufferedReader in2 = new BufferedReader(new InputStreamReader(s2.getInputStream()));
PrintStream out2 = new PrintStream(s2.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str;
while(true){
System.out.println("Waiting for token");
str=in2.readLine();
if(str.equalsIgnoreCase("Token")){
System.out.println("Do you want to send some data");
System.out.println("Enter Yes or No");
str=br.readLine();
if(str.equalsIgnoreCase("Yes")){
System.out.println("Enter the data");
str=br.readLine();
System.out.println(str);
}
System.out.println("Token");
}
}
}
}
```



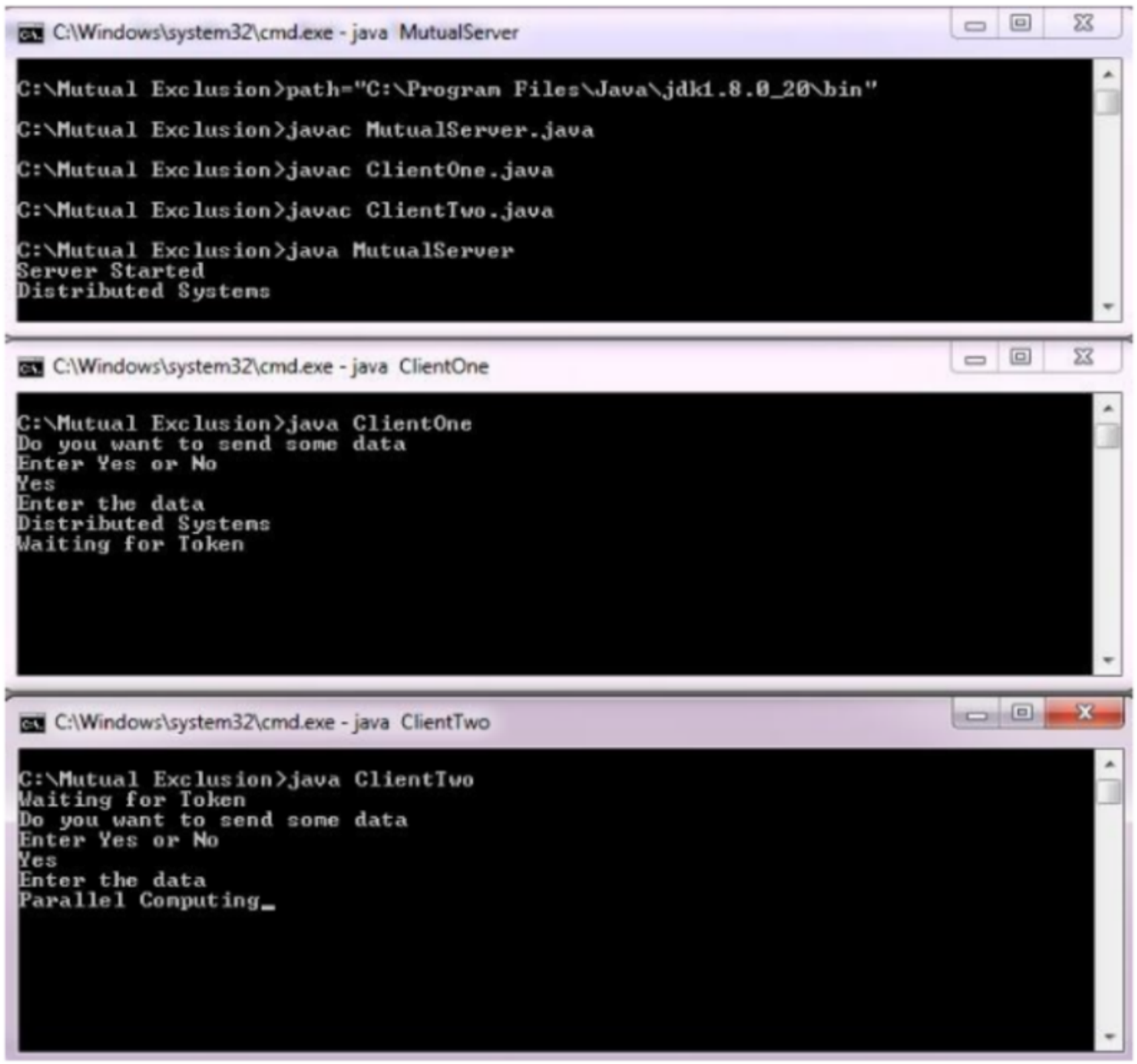
Step 4 – Run Server Program and keep it running till we connect the clients



Step 5 – Open new Command prompt and Run ClientOne Program on it and keep it running till ClientTwo starts.



Step 6 – Open one more Command prompt to Run ClientTwo Program. The output allows both the clients to use token and share their messages with each other using Token Ring. To send the message, the client has to accept the token by typing type Yes followed by the message alternately and has to type No to release the token.



Experiment No. 6

Title: Implementation of Load Balancing approach in distributed environment.

Code:

```
import java.util.Arrays;

public class LoadBalance {

 public static void main(String[] args) {

    int totalServers= Integer.parseInt(args[0]);

    int totalRequests = Integer.parseInt(args[1]);

    int sampleTime = Integer.parseInt(args[2]);

    System.out.println("total Servers: " + totalServers);

    System.out.println("total Reqeusts: " + totalRequests);

    System.out.println("Sample Time: " + sampleTime);

    // Create server queues.

    RandomArrayQueue<Queue<Integer>> servers;

    servers = new RandomArrayQueue<Queue<Integer>>();

    for (int i = 0; i < totalServers; i++) {

    servers.enqueue(new Queue<Integer>());

    }

    // Passing reqeusts to servers

    for (int j = 0; j < totalRequests; j++) {

    Queue<Integer> min = servers.sample();

    for (int k = 1; k < sampleTime; k++) {

    // Pick a sample

    Queue<Integer> queue = servers.sample();

    if (queue.size() < min.size()) min = queue;

    }

    // min surpposed to be the shortest server queue.

    min.enqueue(j);

    }
```

```
int i = 0;
double[] lengths = new double[totalServers];
for (Queue<Integer> queue : servers)
lengths[i++] = queue.size();
System.out.println(Arrays.toString(lengths));
}
}
```

Output:

```
-> java LoadBalance 5 5000 0
total Servers: 5
total Reqeusts: 5000
Sample Time: 0
[956.0, 995.0, 959.0, 1080.0, 1010.0]

-> java LoadBalance 5 5000 2
total Servers: 5
total Reqeusts: 5000
Sample Time: 2
[999.0, 1000.0, 1000.0, 1001.0, 1000.0]
```

Experiment No. 7

Title: Implementation of passive time server centralized algorithm for Synchronization of clock (Cristian's Algorithm).

Code:

```
import java.io.*;

import java.util.*;

import java.net.*;

public class Master

{


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

        {

                String send="",r="";


                Socket MyClient = new Socket("192.168.0.106",25);

                System.out.println("Connected as Master");

                DataInputStream din=new DataInputStream(MyClient.getInputStream());

                DataOutputStream dout = new DataOutputStream(MyClient.getOutputStream());

                Scanner sc = new Scanner(System.in);

                do

                {

                        System.out.print("Message('close' to stop): ");

                        send = sc.nextLine();

                        dout.writeUTF(send);

                        dout.flush();

                }while(!send.equals("stop"));

                dout.close();

                din.close();

                MyClient.close();
```

```
        }
}
```

Slave.java

```
import java.io.*;
import java.util.*;
import java.net.*;
public class Slave
{

        public static void main(String args[])throws Exception
        {
                String r="";
                Socket MyClient = new Socket("192.168.0.106",25);
                System.out.println("Connected as Slave");
                DataInputStream din=new DataInputStream(MyClient.getInputStream());

                do
                {
                        r=din.readUTF();
                        System.out.println("Master says: " + r);
                }while(!r.equals("stop"));

                din.close();
                MyClient.close();
        }
}
```

Server.java

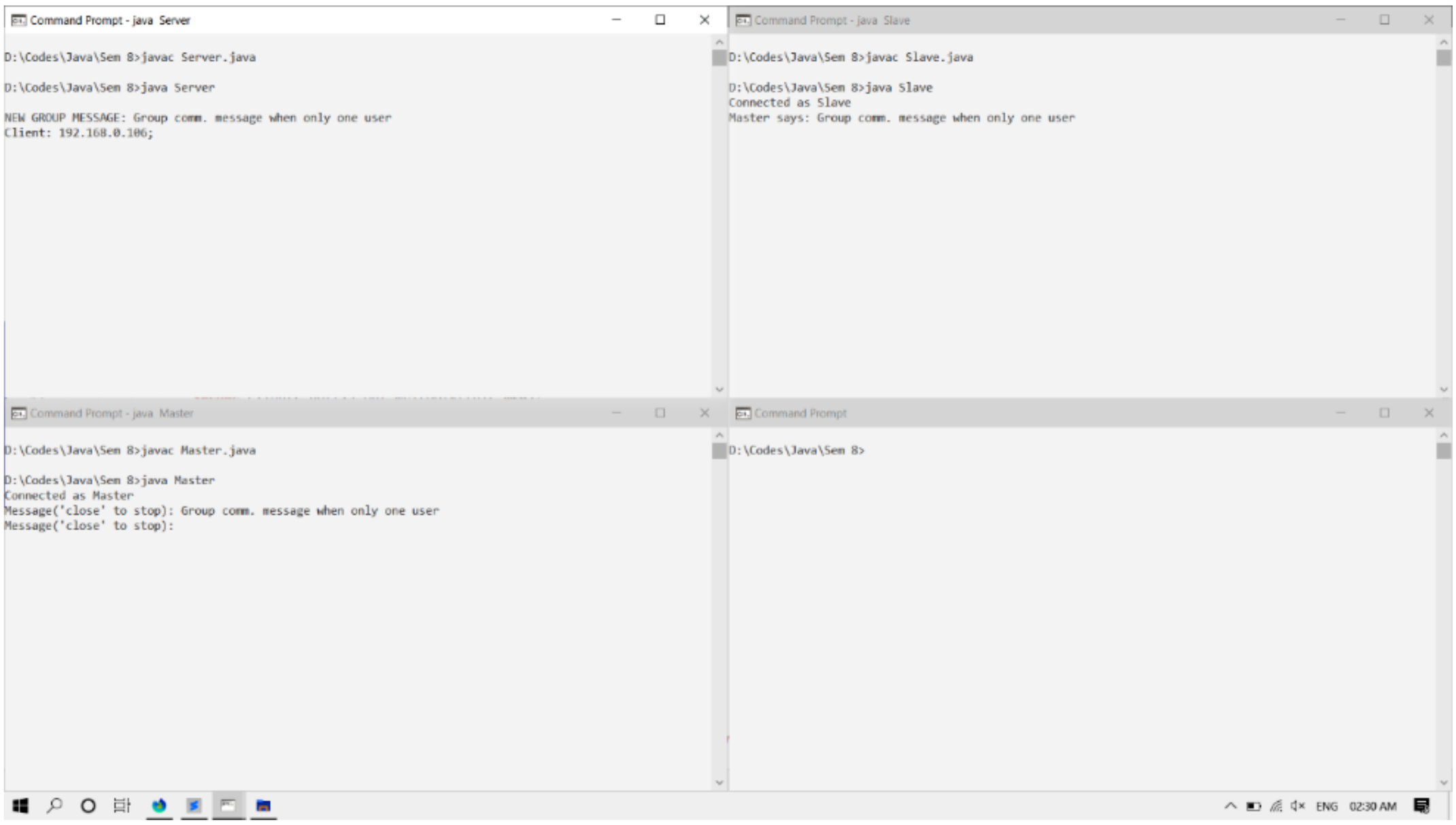
```
import java.util.*;
import java.io.*;
import java.net.*;
public class Server {
    static ArrayList<ClientHandler> clients;
    public static void main(String args[]) throws Exception{
        //Server server = new Server();
        ServerSocket MyServer = new ServerSocket(25);
        clients = new ArrayList<ClientHandler>();
        Socket ss = null;
        Message msg = new Message();
        int count = 0;
        while(true) {
            ss = null;
            try {
                ss = MyServer.accept();
                DataInputStream din =new DataInputStream(ss.getInputStream());
                    DataOutputStream dout=new
DataOutputStream(ss.getOutputStream());
                    ClientHandler chlr = new ClientHandler(ss,din,dout,msg);
                    Thread t = chlr;
                    if (count > 0)
                        clients.add(chlr);
                    count++;
                    //System.out.println(threads);
                    t.start();
            }
            catch(Exception E){
                continue;
```

```
                }
            }
        }
    }
    class Message{
        String msg;
        public void set_msg(String msg){
            this.msg = msg;
        }
        public void get_msg(){
            System.out.println("\nNEW GROUP MESSAGE: " + this.msg);
            for(int i = 0; i < Server.clients.size(); i++){
                try{
                    System.out.print("Client: " + Server.clients.get(i).ip + ";");
                    Server.clients.get(i).out.writeUTF(this.msg);
                    Server.clients.get(i).out.flush();
                }
                catch(Exception e){
                    System.out.print(e);
                }
            }
        }
    }
    class ClientHandler extends Thread{
        DataInputStream in;
        DataOutputStream out;
        Socket socket;
        int sum;
        float res;
```

```
        boolean conn;
        Message msg;
        String ip;
        public ClientHandler(Socket s, DataInputStream din, DataOutputStream dout, Message
msg) {
            this.socket = s;
            this.in = din;
            this.out = dout;
            this.conn = true;
            this.msg = msg;
            this.ip = (((InetSocketAddress)
this.socket.getRemoteSocketAddress()).getAddress()).toString().replace("/","");
        }
        public void run(){
            while(conn == true){
                try{
                    String input = this.in.readUTF();
                    //System.out.println("From host " + this.ip + ':' + input);
                    //String msg = "From host " + this.ip + ':' + input;
                    this.msg.set_msg(input);
                    this.msg.get_msg();
                }
                catch(Exception E){
                    conn = false;
                    System.out.println(E);
                }
            }
            closeConn();
        }
```

```
        public void closeConn(){
                try{
                this.out.close();
                this.in.close();
                this.socket.close();
                }
                catch(Exception E){
                        System.out.println(E);
                }
        }
}
```

Output:



Command Prompt - java Server

```
D:\Codes\Java\Sem 8>javac Server.java

D:\Codes\Java\Sem 8>java Server

NEW GROUP MESSAGE: Group comm. message when only one user
Client: 192.168.0.106;
NEW GROUP MESSAGE: Group comm. message after one nmore slave joined.
Client: 192.168.0.106; Client: 192.168.0.106;
```

Command Prompt - java Slave

```
D:\Codes\Java\Sem 8>javac Slave.java

D:\Codes\Java\Sem 8>java Slave
Connected as Slave
Master says: Group comm. message when only one user
Master says: Group comm. message after one nmore slave joined.
```

Command Prompt - java Master

```
D:\Codes\Java\Sem 8>javac Master.java

D:\Codes\Java\Sem 8>java Master
Connected as Master
Message('close' to stop): Group comm. message when only one user
Message('close' to stop): Group comm. message after one nmore slave joined.
Message('close' to stop):
```

Command Prompt - java Slave

```
D:\Codes\Java\Sem 8>java Slave
Connected as Slave
Master says: Group comm. message after one nmore slave joined.
```

ENG 02:31 AM