LAB MANUAL

Name: SHAH HETVI

Enrollment No.: 180850131016

Class: CSE sem. 7

Subject: Distributed System

Subject Code: 3170719

HJD INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH, KERA
COMPUTER ENGINEERING DEPARTMENT

HJD INSTITUTE OF TECHNICAL
EDUCATION AND RESEARCH, KERA

Approved by AICTE and affiliated to
Gujarat Technological University

CERTIFICATE

This is to certify that **Miss SHAH HETVI Enrollment No 180850131016** of programme **BE 7th SEM Computer Engineering** has satisfactorily completed **her** term work in **Distributed System(3170719)** for the term ending in 2021-2022.

Ms. Pooja Gunsai
(Lab In-charge)

Mr. Vishal Bhimani
(Head Of Department)

INDEX

Practical-1

Aim: Write a program for communication between client and server in which server reply the same message in capitals using UDP.

Client Program:

```
import java.io.*;

import java.net.*;


class UDPClient  {

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

    BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));

    DatagramSocket clientSocket = new DatagramSocket();

    InetAddress IPAddress = InetAddress.getByName("localhost");

    byte[] sendData = new byte[1024];

    byte[] receiveData = new byte[1024];

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

    String sentence11=inFromUser.readLine();

    System.out.println("Roll no:");

    String sentence12=inFromUser.readLine();

    System.out.println("INPUT FOR PROGRAM:");

    String sentence = inFromUser.readLine();

    sendData = sentence.getBytes();

    DatagramPacket sendPacket = new DatagramPacket(sendData,    sendData.length,IPAddress,
6789);

    clientSocket.send(sendPacket);
```

```
    DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence = new String(receivePacket.getData());
    System.out.println("FROM SERVER:" + modifiedSentence);
    clientSocket.close();
  }
}
```

Server Program:

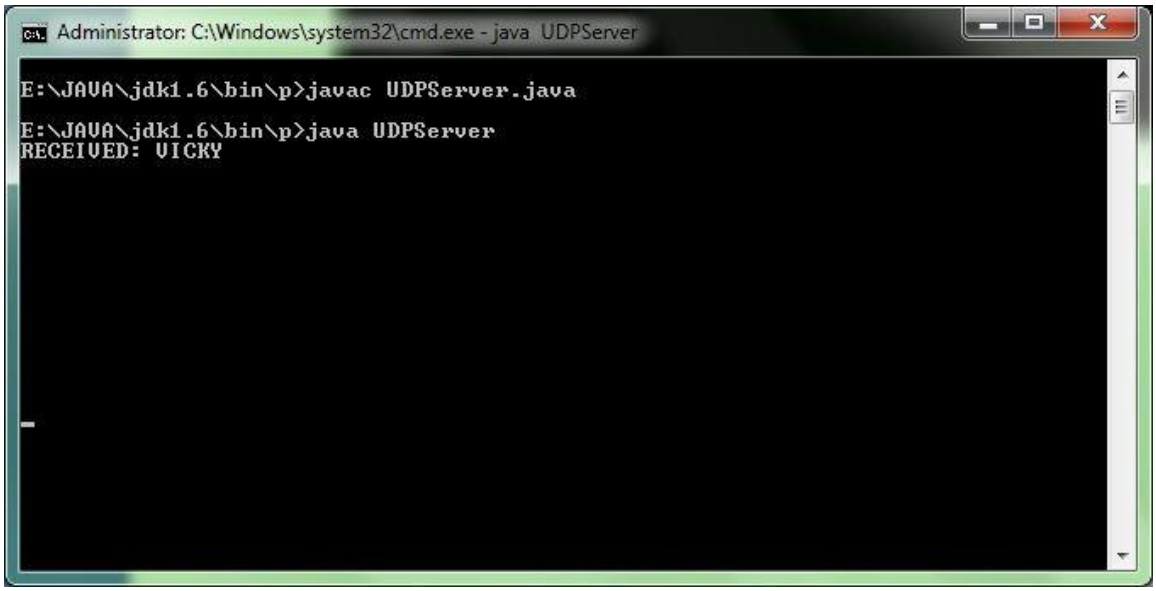
```
import java.io.*;
import java.net.*;

class UDPServer{
  public static void main(String args[]) throws Exception{
      DatagramSocket serverSocket = new DatagramSocket(6789);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while(true)
          {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String( receivePacket.getData());
            System.out.println("RECEIVED: " + sentence);
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();String
            capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
```

```
            DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }}
```

output:

output of client:

Output of server:



Practical-2

Aim: Write a program for communication between client and server in which server reply the same message in capitals using TCP.

Client Program:

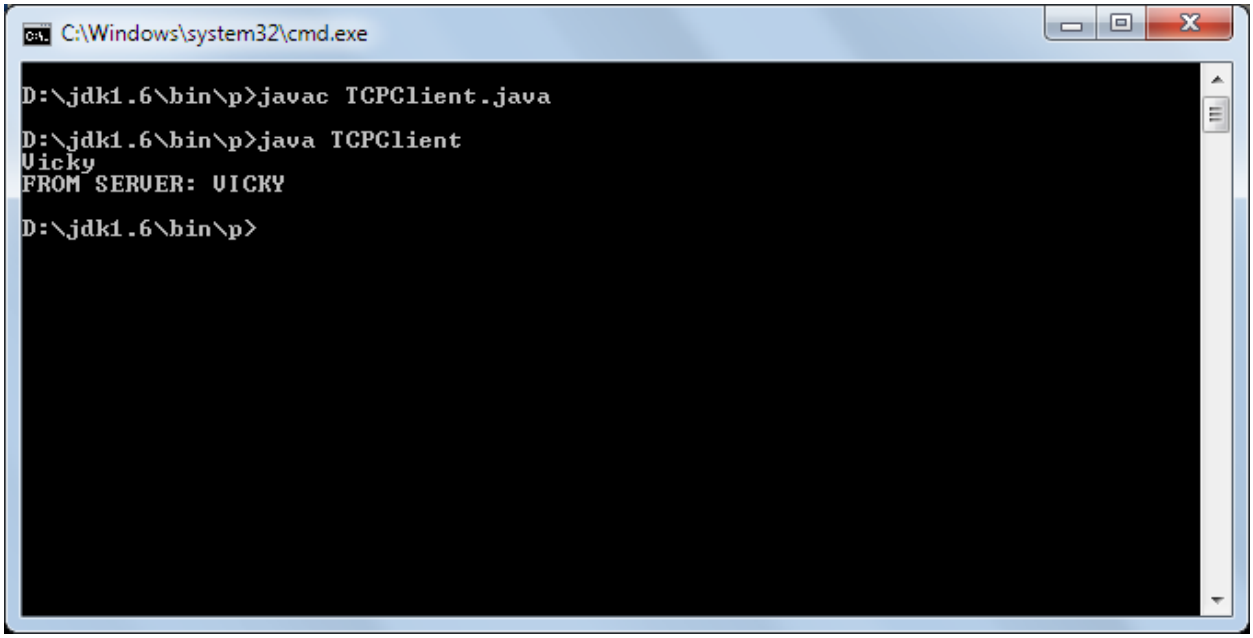
```
import java.io.*;

import java.net.*;

class TCPClient{

 public static void main(String argv[]) throws Exception {

  String sentence;

  String modifiedSentence;

  BufferedReader inFromUser = new BufferedReader( new
InputStreamReader(System.in));

  Socket clientSocket = new Socket("localhost", 6789);

  DataOutputStream outToServer = new
DataOutputStream(clientSocket.getOutputStream());

  BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

  sentence = inFromUser.readLine();

  outToServer.writeBytes(sentence + '\n');

  modifiedSentence = inFromServer.readLine();

  System.out.println("FROM SERVER: " + modifiedSentence);

  clientSocket.close();

 }          }
```

Server Program:

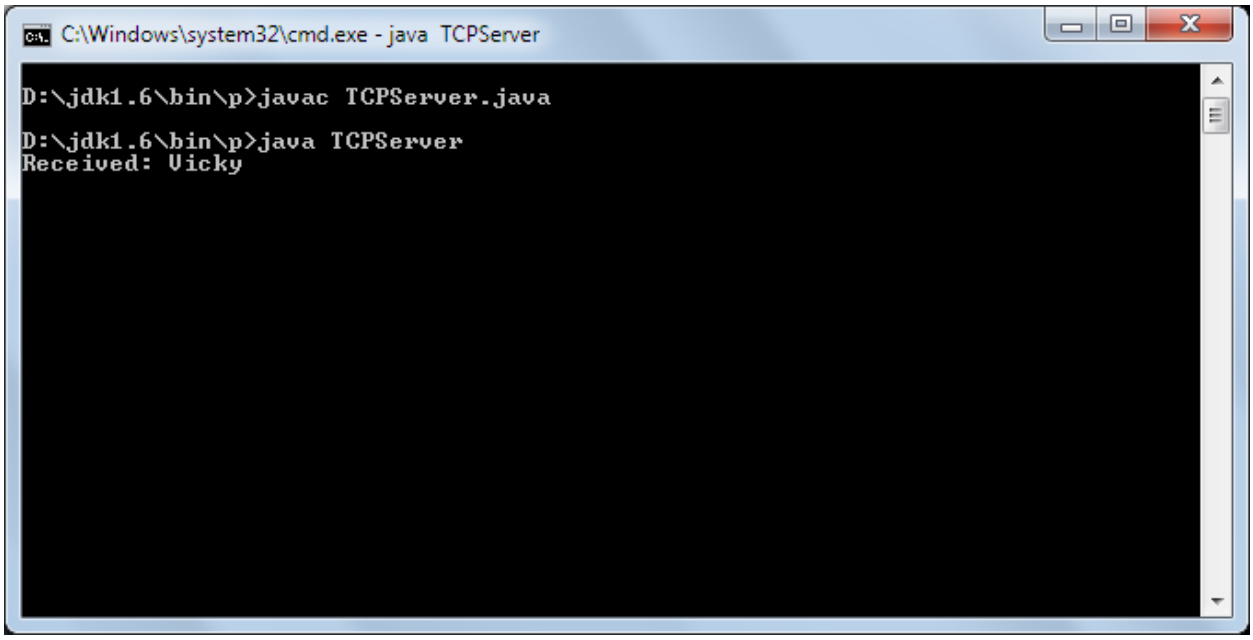
```
import java.io.*;
import java.net.*;
class TCPServer{

  public static void main(String argv[]) throws Exception{
      String clientSentence;
      String capitalizedSentence;
      ServerSocket welcomeSocket = new ServerSocket(6789);
      while(true){

        Socket connectionSocket = welcomeSocket.accept();
        BufferedReader inFromClient =
          new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
        DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());
        clientSentence = inFromClient.readLine();
        System.out.println("Received: " + clientSentence);
        capitalizedSentence = clientSentence.toUpperCase() + '\n';
        outToClient.writeBytes(capitalizedSentence);
            }
          }
    }
```

output:

output of client:



Output of Server:



PRACTICAL -3

Aim:Write a java program in for communication between client and server where server is calculator.

Client Program:

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

class calclient{
        public static void main(String args[]){
        Socket s=null;
        ObjectInputStream ois=null;
        ObjectOutputStream oos=null;
        Scanner sc=new Scanner(System.in);
        try{

        s=new Socket("localhost",1444);
        System.out.println("connected");
      System.out.println("NAME:");
      String n=sc.next();
      System.out.println("CALCULATOR");
      ois=new ObjectInputStream(s.getInputStream());
      String d=(String)ois.readObject();
        oos=new ObjectOutputStream(s.getOutputStream());
        System.out.println((String)ois.readObject());
```

```
        oos.writeObject(sc.next());
        System.out.println((String)ois.readObject());
        oos.writeObject(sc.next());
        System.out.println((String)ois.readObject());
        oos.writeObject(sc.next());
        System.out.println("answer="+(String)ois.readObject());
        }
        catch(Exception e){
        System.out.println("error"+e.getMessage());
                }
        }
}
```

Server Program :

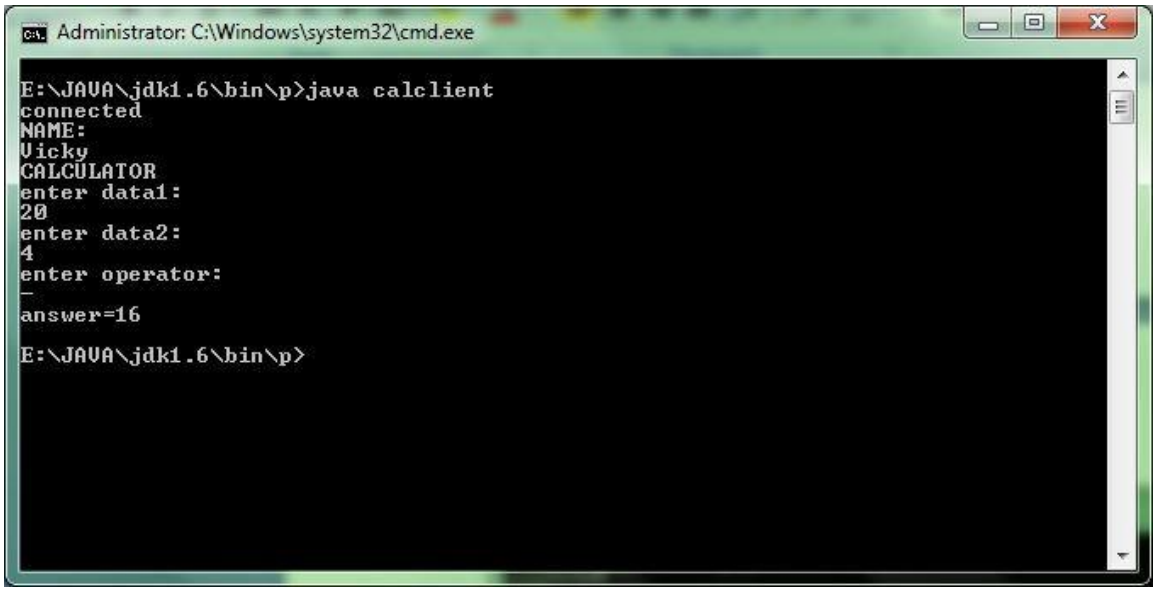
```
import java.net.*;
import java.util.*;

class calserver{
        public static void main(String args[]){
         ServerSocket ss=null;
        Socket s=null;
        ObjectOutputStream oos=null;
        ObjectInputStream ois=null;
        int d1=0,d2=0;
        String op;
        try{
```

```
 ss=new ServerSocket(1444);
 System.out.println("server is created");
 }
catch(IOException ioe){
        System.out.println(ioe.getMessage());
}
while(true){
         System.out.println("waiting for connection");
         try{
         s=ss.accept();
         System.out.println("connected");
         oos=new ObjectOutputStream(s.getOutputStream());
         oos.writeObject("welcome client");
         ois=new ObjectInputStream(s.getInputStream());
         oos.writeObject("enter data1:");
         d1=Integer.parseInt((String)ois.readObject());
         oos.writeObject("enter data2:");
         d2=Integer.parseInt((String)ois.readObject());
         oos.writeObject("enter operator:");
         op=(String)ois.readObject();
         if(op.equals("+"))
         oos.writeObject((d1+d2)+"");
         else if(op.equals("-"))
         oos.writeObject((d1-d2)+"");
        else if(op.equals("*"))oos.writeObject((d1*d2)+""
```

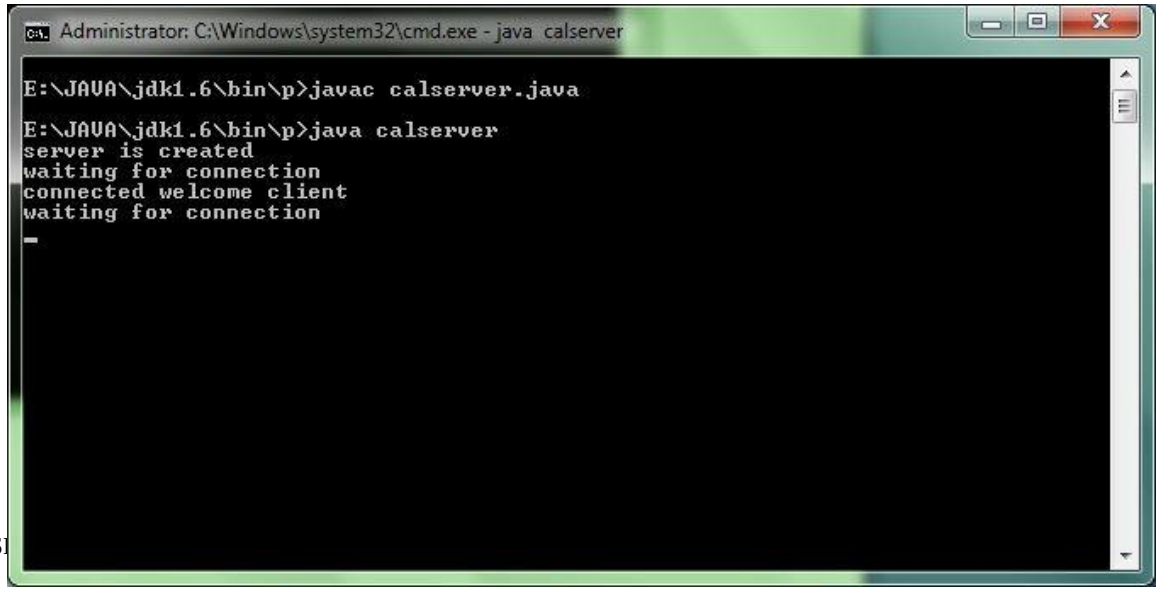
```
else if(op.equals("/"))
oos.writeObject((d1/d2)+"");
        }
        catch(Exception e){
        System.out.println(e.getMessage());
        }}}     }
```

Output:

Output of Client:



Output of Server:



Practical-4

Aim: Write a program for "Hello" using RMI

Stub Interface: method.java

```
import java.rmi.*;
interface demo extends Remote{
public String name()throws RemoteException;
}
```

Server Program: server_rmi1.java

```
import java.rmi.*;
importjava.rmi.registr*
importjava.rmi.server.*;
class server_rmi1 implements demo{
public server_rmi1()
{       super();       }
String message;
public server_rmi1(String msg){
message=msg;
}
public String name(){
return "hello";
}
public static void main(String args[]){
try{
demo d=new server_rmi1("Sumit");
Demostub=(demo)UnicastRemoteObject.exportObject(0
```

)

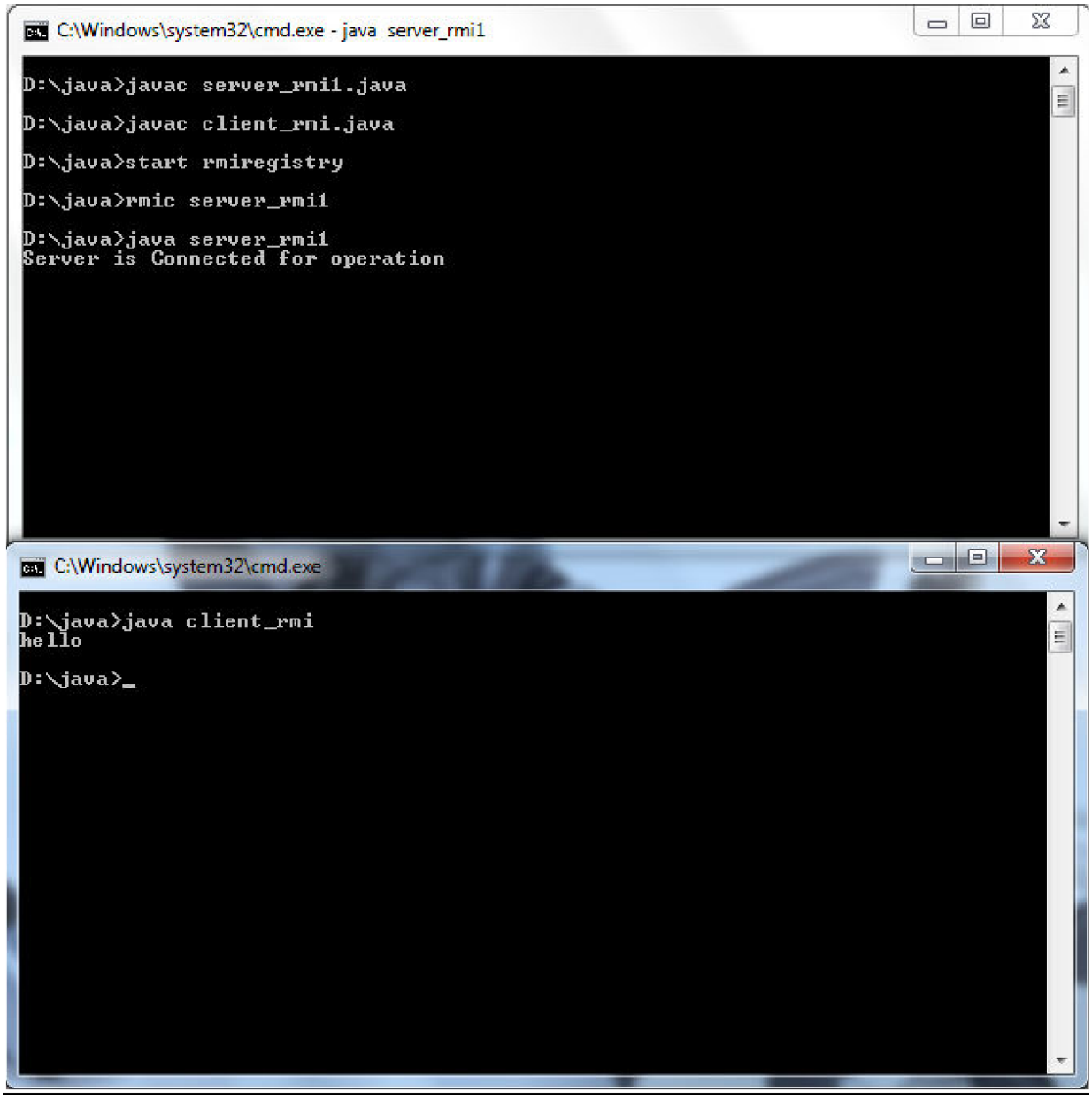
```
Registry r=LocateRegistry.getRegistry();
r.rebind("hi",stub);
System.out.println("Server is Connected for operation");
}
catch(Exception e){
System.out.println(e);
}
}
}
```

Client Program: client_rmi.java

```
import java.rmi.*;
importjava.rmi.registr*;
class client_rmi{
public static void main(String args[]) {
Try {
Registry r=LocateRegistry.getRegistry(); demo d=(demo)r.lookup("hi");
String message=d.name();
System.out.println(message
)
}
catch(Exception e){
System.out.println(e);
}
}
```

```
}
```

Output:


C:\Windows\system32\cmd.exe - java server_rmi1
D:\java>javac server_rmi1.java
D:\java>javac client_rmi.java
D:\java>start rmiregistry
D:\java>rmic server_rmi1
D:\java>java server_rmi1
Server is Connected for operation
C:\Windows\system32\cmd.exe
D:\java>java client_rmi
hello
D:\java>_

Practical-5

Aim: Write a program for fetching Server Time using RMI

Stub Interface: method.java

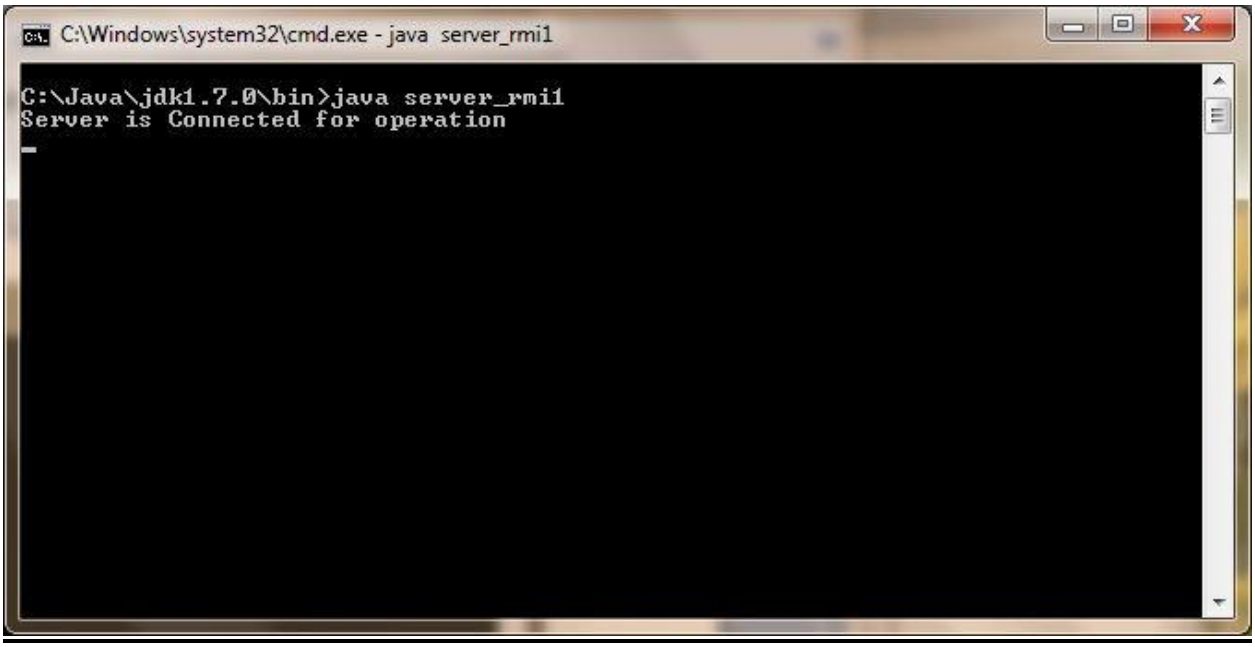
```
import java.rmi.*;
interface demo extends Remote{
public String name()throws RemoteException;
}
```

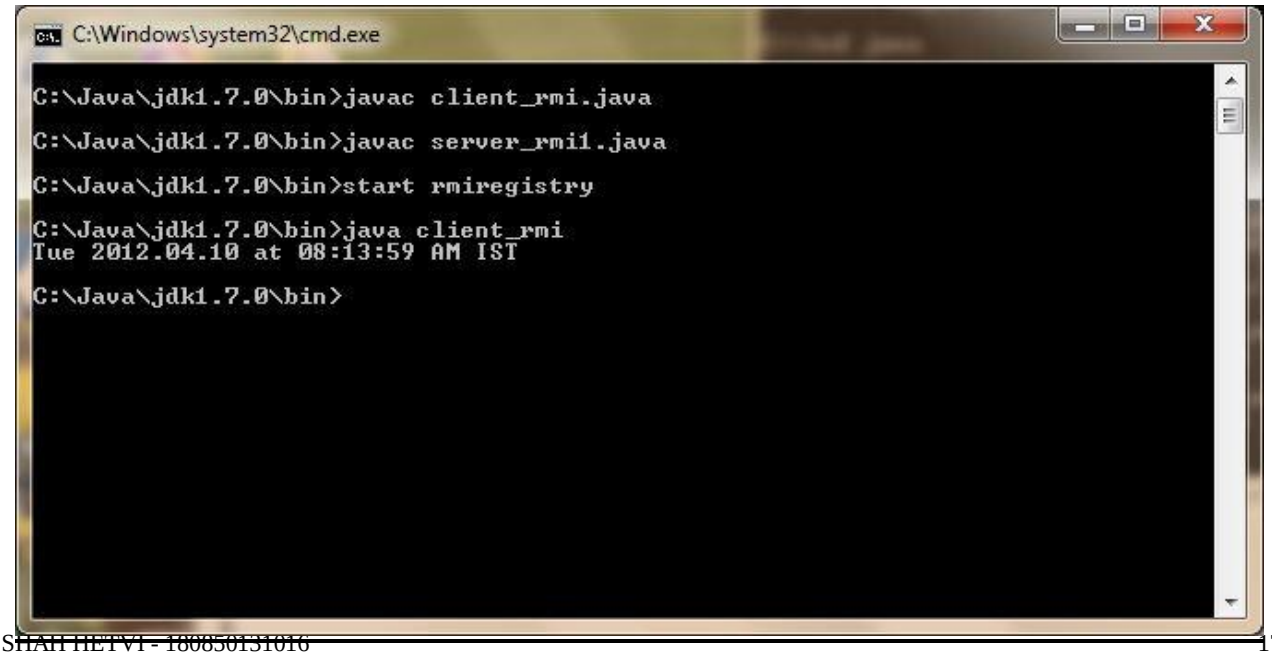
Server Program: server_rmi1.java

```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.util.*;
import java.lang.*;
import java.text.*;
class server_rmi1 implements demo{
public server_rmi1()
{ super(); }
String message;
Date date=new Date();
SimpleDateFormat ft=new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
String t= ft.format(date);
Registry r=LocateRegistry.getRegistry();
demo d=(demo)r.lookup("hi");
String message=d.name();
```

```
System.out.println(message);

}

catch(Exception e){

System.out.println(e);
}}}
```

Output:





Practical-6

Aim: Write a program for RPC Implementation.

Stub File :

add.x

```
struct intpair {
        int a;
        int b;
};
program ADD_PROG {
        version ADD_VERS {
                int ADD(intpair) = 1;
        } = 1;
} = 234;
```

Client File: add_clnt

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <memory.h>       /* for memset */
#include "add.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

int *
add_1(intpair *argp, CLIENT *clnt)
{
        static int clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, ADD,
```

```
            (xdrproc_t) xdr_intpair, (caddr_t) argp,
            (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
            TIMEOUT) != RPC_SUCCESS) {
            return (NULL);
        }
        return (&clnt_res);}
```

Server File: add_server

```
/*
 * This is sample code generated by rpcgen.
 */

#include "add.h"

int *
add_1_svc(intpair *argp, struct svc_req *rqstp)
{
        static int  result;

        printf("Hello world\n");
        return &result;
}
```

Guidelines for Run Program:

1.rpcgen -C add.x:

-The -C flag (upper-case C) tells rpcgen to generate C code that conforms to ANSI C.

2.rpcgen -a -C add.x

-This will create the client (add_client.c), the server (add_server.c), and the makefile (makefile.add or Makefile.add, depending on whether you are using SunOS or Linux; OS X will not create a makefile).

3. Insert code in "add_server.c"

-

4. Open -> "MakeFile.add" and change -> "CFLAGS += -g -DRPC_SVC_FG" and "RPCGENFLAGS = -C"

5. Compile: make -f Makefile.add

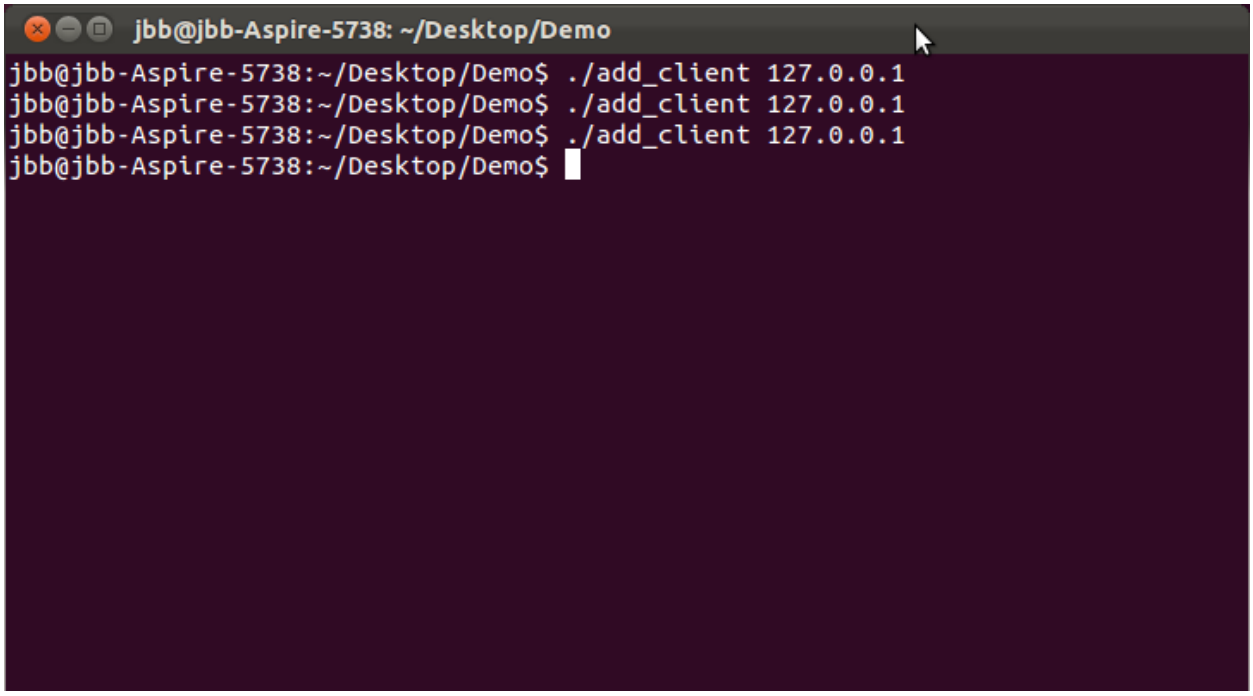
-This will allow us to simply type make with no parameters to recompile since by default make looks for a file named makefile or Makefile.

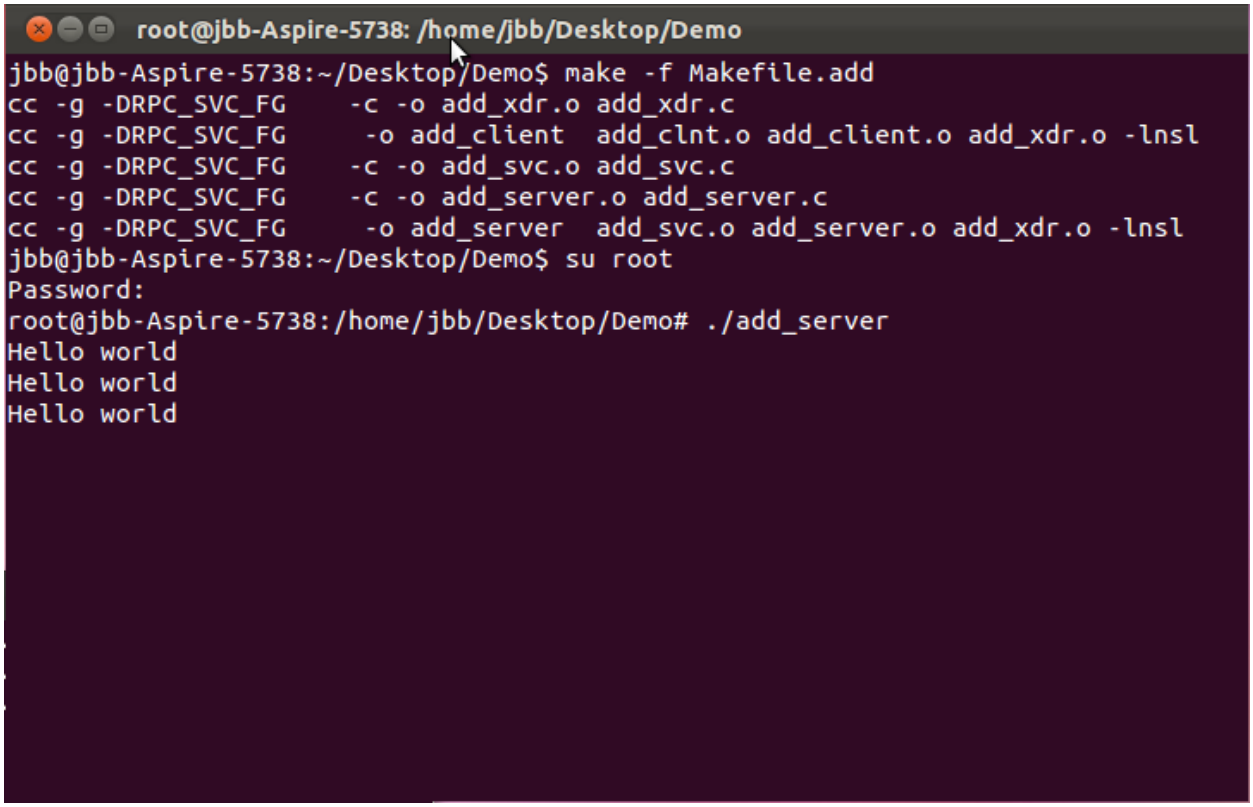
6. Check Portmapper: rpcinfo -p

7.Run Server: ./add_server

8.Run Client: ./add_client 127.0.0.1

Output:


jbb@jbb-Aspire-5738: ~/Desktop/Demo
jbb@jbb-Aspire-5738:~/Desktop/Demo$ ./add_client 127.0.0.1
jbb@jbb-Aspire-5738:~/Desktop/Demo$ ./add_client 127.0.0.1
jbb@jbb-Aspire-5738:~/Desktop/Demo$ ./add_client 127.0.0.1
jbb@jbb-Aspire-5738:~/Desktop/Demo$


root@jbb-Aspire-5738: /home/jbb/Desktop/Demo
jbb@jbb-Aspire-5738:~/Desktop/Demo$ make -f Makefile.add
cc -g -DRPC_SVC_FG -c -o add_xdr.o add_xdr.c
cc -g -DRPC_SVC_FG -o add_client add_clnt.o add_client.o add_xdr.o -lnsl
cc -g -DRPC_SVC_FG -c -o add_svc.o add_svc.c
cc -g -DRPC_SVC_FG -c -o add_server.o add_server.c
cc -g -DRPC_SVC_FG -o add_server add_svc.o add_server.o add_xdr.o -lnsl
jbb@jbb-Aspire-5738:~/Desktop/Demo$ su root
Password:
root@jbb-Aspire-5738:/home/jbb/Desktop/Demo# ./add_server
Hello world
Hello world
Hello world

PRACTICAL -7

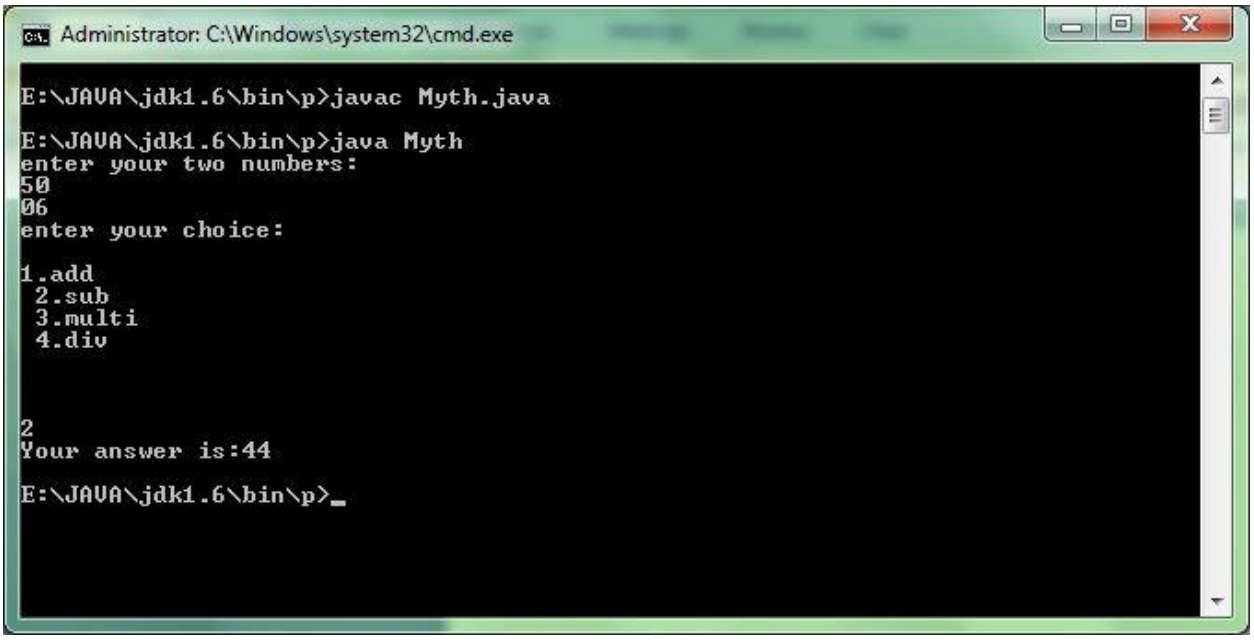
***Aim*: Write a program for Simple Calculator Using Thread**

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


class chth implements Runnable{
Thread t;
int n1,n2;
int op;

chth(int x,int y,int opp){
t=new Thread(this);
n1=x; n2=y; op=opp;
t.start();
}
public void run(){
   int ans;
   if(op==1)
   ans=n1+n2;
  else if(op==2)
  ans=n1- n2;
   else if(op==3)
   ans=n1*n2;
   else
  ans= n1/n2 ;
```

```
System.out.println("Your answer is:"+ans);
try
 { Thread.sleep(1000); }
 catch(Exception e) {  }
 }    }
 public class Myth{
  public static void main(String args[]) throws IOException{
 BufferedReader br=new BufferedReader (new InputStreamReader ( System.in));
 System.out.println("enter your name:");
 String str1=br.readLine();
 System.out.println("enter your two numbers:");
 BufferedReader br=new BufferedReader (new InputStreamReader ( System.in));
 String str,ar;
 int a,b,c;
 str=br.readLine();a=Integer.parseInt(str);
 ar=br.readLine();
 b=Integer.parseInt(ar);
 System.out.println("enter your choice:");
 System.out.println(" \n1.add \n 2.sub \n 3.multi \n 4.div\n\n\n");
 String sr=br.readLine();
  c=Integer.parseInt(sr);
 try { Thread.sleep(1000); }
 catch(Exception e) { }
  switch(c){
```

```
case 1:
chth c1=new chth(a,b,1); break;case 2:
   chth c2=new chth(a,b,2);
   break;
case 3:
   chth c3=new chth(a,b,3);
    break;
case 4:
   chth c4=new chth(a,b,4);
   break;
  default:System.out.println("wrong choice");
}
}   }
```

OUTPUT:



Practical-8

***Aim*: Write a program for implementing a Bank account using ThreadSafe.**

Program: threadsf.java

```
import java.io.*;
class account{
int bal=10000;
void
read(Stringth) {

System.out.println("Checking your
Balance..."); System.out.println("\n your
Balance is:"+bal+"\n");
}

void edit(String th)throws IOException{
System.out.println("\n"+th+" has edit to data");
System.out.println(" 1.credit \n 2.debit");
System.out.println("Enter your choice:");
BufferedReader br=new BufferedReader (new InputStreamReader (
System.in)); String a=br.readLine();
int ch=Integer.parseInt(a);
 System.out.println("Enter your amount:");

String b=br.readLine();
int amt=Integer.parseInt(b); try {
Thread.sleep(1000); }
catch(Exception e) { }
```

```
public void run(){
 int c;
 String sr;
 synchronized(act)
 {       try {
             BufferedReader br=new BufferedReader (new InputStreamReader ( System.in));
             System.out.println("\n"+thn+" Start");
              System.out.println(" 1.check Balance \n 2.edit
              account "); System.out.println("Enter your choice:");

             sr = br.readLine();
             c=Integer.parseInt(sr);try {
             Thread.sleep(1000); }
             catch(Exception e) { }
             switch(c){
             case 1:
                   act.read(thn);
                   try { t.sleep(2000);}
             catch (InterruptedException ex) { } break;case 2:
                   act.read(thn);
                    try  {   act.edit(thn);   }
                   catch(Exception e) {  }
                      break;
             default:
   System.out.println("Wrong choice");}}
   catch (IOException ex) {  }
                        } } }
```

```
class threadsf{
public static void main(String [] args){
account act=new account();
BufferedReader br=new BufferedReader (new InputStreamReader (
System.in));
 try  {
        System.out.println("enter your name:");
        String str1=br.readLine();
}
catch(Exception e) {  };

user u1=new
user("ThreadA",act);
  user u2=new
user("ThreadB",act);
}
 }
```

Outputs:

Case 1: If the Balance or amount received is LessThan 100.

Case 2: All Normal Transaction Done.

PRACTICAL 10

AIM:- A simple CORBA implementation using Java

Instructions

1. Write the IDL file

The IDL file defines the interface that will be used by the client and server for communicating and passing objects.
When the IDL file gets compiled, it will produce a number of files, known as the stub and skeleton:

- The stub is used by the client to communicate with the server
- The skeleton is used by the server to communicate with the client
- The stub and skeleton communicate with the ORB server to facilitate the remote procedure call

The module in the IDL file will correspond to the package and directory in which the Java code will be generated

Echo.idl

```
module EchoApp {
   interface Echo {
      string echoString();
   };
};
```

2. Generate the stub and skeleton code

There is an idlj program that comes with the JDK for generating the stub and skeleton code in Java

idlj –fall sum.idl
The following files are generated by the idlj program:

- _EchoStub.java
- Echo.java
- EchoHelper.java
- EchoHolder.java
- EchoOperations.java
- EchoPOA.java

3. Write the server code

The server program will inherit from the EchoPOA class that was generated as part of the idlj program.
The EchoPOA class implements the EchoOperations interface

- This interface contains the methods we defined in the Echo.idl file, but standardized to Java.

We create an EchoServer.java class that extends the abstract EchoPoa class and then implement the methods contained in it.We create a main method in Server.java to communicate with the object request broker (ORB), registering the server with the ORB so clients are able to find it.

4. Write the client code

The client program Client.java needs to get a reference to the ORB then resolve the name of the server object it would like to invoke.

- This is ECHO-SERVER in this case

After getting an instance of a Server object from the server, it can invoke methods on it just like a method within its own JVM.

5. Compile the code

1. Compile the stub and skeleton from the directory that contains the IDL file.

   Windows

   javac EchoApp*.java

   Linux

   javac EchoApp/*.java

2. Generate a JAR file from the compiled stub and skeleton.

   Windows

   jar cvf echoapp.jar EchoApp*.class

   Linux

   jar cvf echoapp.jar EchoApp/*.class

3. Compile the server and client classes

   Windows

   javac -classpath .;echoapp.jar Server.java EchoServer.java Client.java

   Linux

   javac -classpath .:echoapp.jar Server.java EchoServer.java Client.java

6. Running the application

1. Start the ORB server

   orbd -ORBInitialPort 1050 -ORBInitialHost localhost

2. Start the server application

   java Server -ORBInitialPort 1050 -ORBInitialHost localhost

3. Start the client application

   java Client -ORBInitialPort 1050 -ORBInitialHost localhost

If everything was compiled correctly the output should be:

Hello World!!!