LATE BHAUSAHEB HIRAY S.S. TRUST'S INSTITUTE OF COMPUTER APPLICATION

ISO 9001-2008 CERTIFIED

S.N. 341, Next to New English School, Govt. Colony, Bandra (East),
Mumbai – 400051, Tel: 91-22-26570892/3181

Open Source System for ADC Lab

CERTIFICATE

The experiments duly signed in this journal represent the bonafide work by default Roll No. of Division in SEMESTER-V of Third Year of the Master in Computer Application (MCA) In the COMPUTER LABORATORY of this College during the session of 2018-19.

Lecturer In-Charge **Head, Department of MCA**

External Examiner

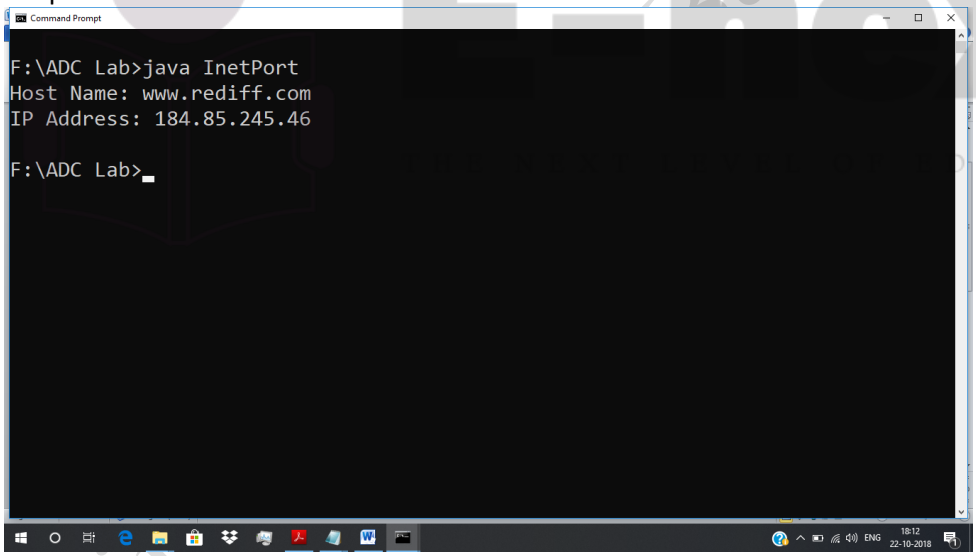
INDEX

Prog. 1 : To Display Port and Host Information

Code :

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

public class InetPort
{
  public static void main(String args[])
  {
     try{
       InetAddress ipadd = InetAddress.getByName("www.rediff.com");
     System.out.println("Host Name: "+ipadd.getHostName());
    System.out.println("IP Address: "+ipadd.getHostAddress());
   URL url = new URL("www.google.com");
 System.out.println("Host Name: "+url.getHost());
System.out.println("Port: "+url.getPort());
      }catch(Exception e)
      { }
  }
}
```

Output:



Prog. 2 Simple Client – Server Communication Program

Code :

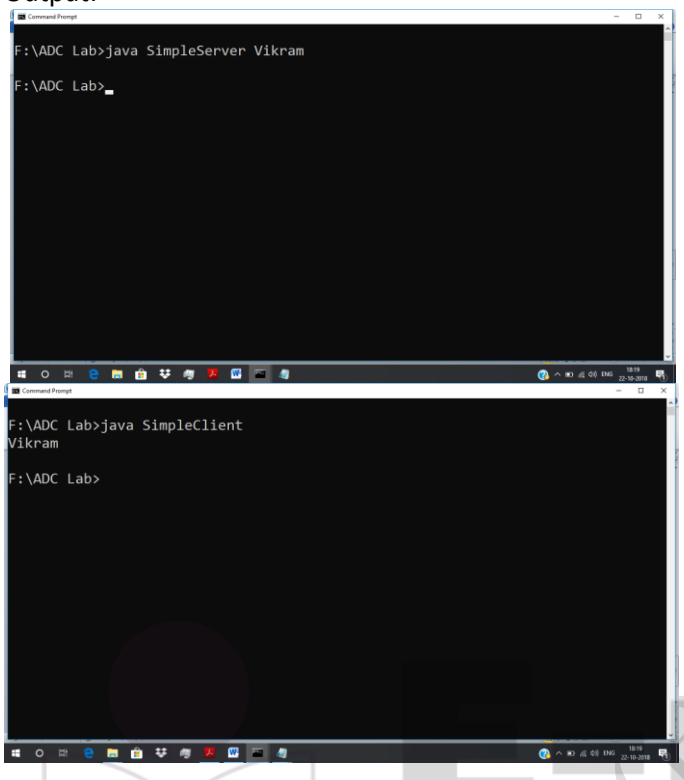
SimpleServer.java

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

public class SimpleServer
{
  public static void main(String args[]) throws Exception
 {
   ServerSocket ss = new ServerSocket(1234);
  Socket s1 = ss.accept();
  OutputStream out = s1.getOutputStream();
  DataOutputStream dos = new DataOutputStream(out);
   dos.writeUTF(args[0]);
   dos.close();
   out.close();
   s1.close();
 }
}
```

SimpelClient.java

```
import java.net.*;
import java.io.*;
public class SimpleClient
{
 public static void main(String args[]) throws Exception
{
   Socket s1 = new Socket("127.0.0.34",1234);
 InputStream is = s1.getInputStream();
 DataInputStream dis = new DataInputStream(is);
 String s = new String(dis.readUTF());
  System.out.println(s);
  dis.close();
  is.close();
s1.close();
}
}
```

Output:



Prog. 3 : Simple Client – Server Connection Program

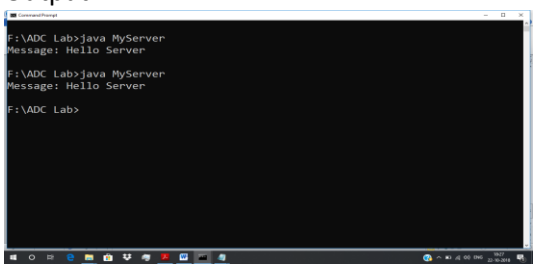
MyServer.java

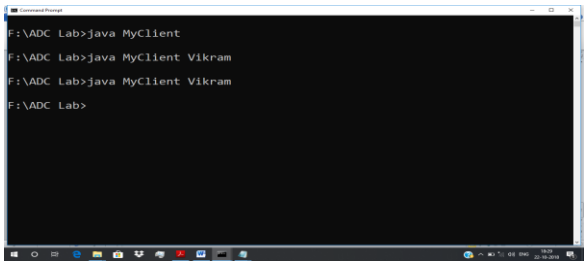
```
import java.io.*;
import java.net.*;
public class MyServer
{
 public static void main(String args[])
{
  try
  {
  ServerSocket ss = new ServerSocket(6666);
  Socket s = ss.accept();
  DataInputStream dis = new DataInputStream(s.getInputStream());
  String str = (String)dis.readUTF();
  System.out.println("Message: "+str);
  ss.close();
  }catch(Exception e)
  { System.out.println(e); }
  }
}
```

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient
{
  public static void main(String args[])
 {
  try
  {
  Socket s = new Socket("localhost",6666);
  DataOutputStream dos = new DataOutputStream(s.getOutputStream());
dos.writeUTF("Hello Server");
dos.flush();
 dos.close();
 s.close();
}catch(Exception e){ }
}
}
```

Output:





Prog. 4 Simple Client – Server Communication for Arithmetic Operation

Code :
ArithTcpServer.java

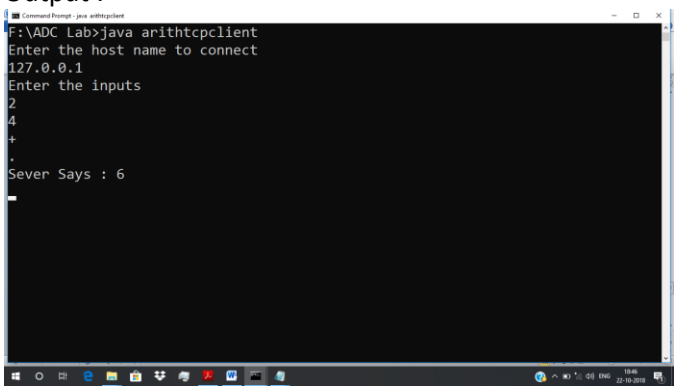
```
import java.io.*;
import java.net.*;
public class arithtcpserver
{
public static void main(String arg []) throws Exception
{
ServerSocket serversoc=new ServerSocket(9999);
Socket clientsoc = serversoc.accept();
PrintWriter out = new PrintWriter(clientsoc.getOutputStream(), true);
BufferedReader in = new BufferedReader(new
InputStreamReader(clientsoc.getInputStream()));
String inputline;
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
try
{
while (true)
{
String s, op=" ", st;
int i=0,c=0;
int [] a=new int[2];
while(true)
{
s=in.readLine();
if(s.equals("+") || s.equals("-") || s.equals("*") || s.equals("/"))
op=s;
else if(s.equals("."))
break;
else
{
a[i]=Integer.parseInt(s);
i++;
}
}
if(op.equals("+"))
c=(a[0]+a[1]);
else if(op.equals("-"))
c=(a[0]-a[1]);
else if(op.equals("*"))
c=(a[0]*a[1]);
else if(op.equals("/"))
c=(a[0]/a[1]);
s=Integer.toString(c);
out.println(s);
}
```

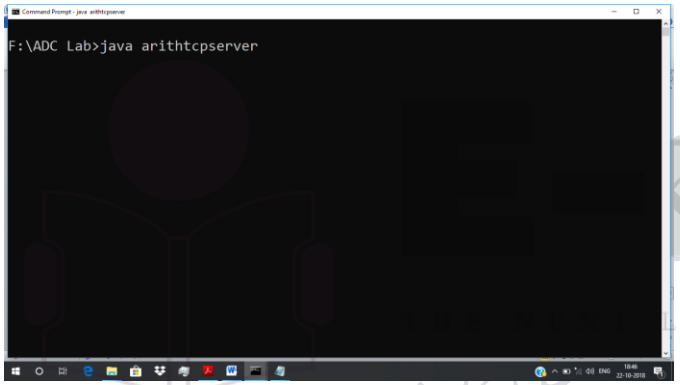
```
}
catch(Exception e)
{
System.exit(0);
}
}
}
```

ArithTcpClient.java

```
import java.io.*;
import java.net.*;
public class arithtcpclient
{
public static void main(String[] args) throws IOException
{
System.out.println("Enter the host name to connect");
String str;
DataInputStream inp=new DataInputStream(System.in);
str=inp.readLine();
Socket clientsoc = new Socket(InetAddress.getLocalHost(), 9999);
System.out.println("Enter the inputs");
PrintWriter out = new PrintWriter(clientsoc.getOutputStream(), true);
BufferedReader in = new BufferedReader(new
InputStreamReader(clientsoc.getInputStream()));
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
String userinput;
try
{
while (true)
{
do
{
userinput = stdin.readLine();
out.println(userinput);
}while(!userinput.equals("."));
System.out.println("Sever Says : " + in.readLine());
}
}
catch(Exception e)
{
System.exit(0);
}
}
}
```

Output :





Prog 5: Simple GUI Based Server – Client Chatting program

Server_Chat.java

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

public class Server_Chat
{
   static Vector ClientSockets;
   static Vector LoginNames;

   public Server_Chat() throws Exception
   {
      ServerSocket soc=new ServerSocket(3000);
      ClientSockets=new Vector();
      LoginNames=new Vector();

      while(true)
      {
         Socket CSoc=soc.accept();
         AcceptClient obClient=new AcceptClient(CSoc);
      }
   }
   public static void main(String args[]) throws Exception
   {

      Server_Chat ob=new Server_Chat();
   }

class AcceptClient extends Thread
{
   Socket ClientSocket;
   DataInputStream din;
   DataOutputStream dout;
   AcceptClient (Socket CSoc) throws Exception
   {
      ClientSocket=CSoc;

      din=new DataInputStream(ClientSocket.getInputStream());
      dout=new DataOutputStream(ClientSocket.getOutputStream());

      String LoginName=din.readUTF();

      System.out.println("User Logged In :" + LoginName);
      LoginNames.add(LoginName);
      ClientSockets.add(ClientSocket);
      start();
```

```
}

public void run()
{
   while(true)
   {

      try
      {
         String msgFromClient=new String();
         msgFromClient=din.readUTF();
         StringTokenizer st=new StringTokenizer(msgFromClient);
         String Sendto=st.nextToken();
         String MsgType=st.nextToken();
         int iCount=0;

         if(MsgType.equals("LOGOUT"))
         {
            for(iCount=0;iCount<LoginNames.size();iCount++)
            {
               if(LoginNames.elementAt(iCount).equals(Sendto))
               {
                  LoginNames.removeElementAt(iCount);
                  ClientSockets.removeElementAt(iCount);
                  System.out.println("User " + Sendto +" Logged Out ...");
                  break;
               }
            }

         }
         else
         {
            String msg="";
            while(st.hasMoreTokens())
            {
               msg=msg+" " +st.nextToken();
            }
            for(iCount=0;iCount<LoginNames.size();iCount++)
            {
               if(LoginNames.elementAt(iCount).equals(Sendto))
               {
                  Socket tSoc=(Socket)ClientSockets.elementAt(iCount);
                  DataOutputStream tdout=new DataOutputStream(tSoc.getOutputStream());
                  tdout.writeUTF(msg);
                  break;
               }
            }
            if(iCount==LoginNames.size())
            {
```

```
                dout.writeUTF("I am offline");
              }
              else
              {

              }
            }
            if(MsgType.equals("LOGOUT"))
            {
              break;
            }

          }
          catch(Exception ex)
          {
            ex.printStackTrace();
          }



      }
    }
}
}
```

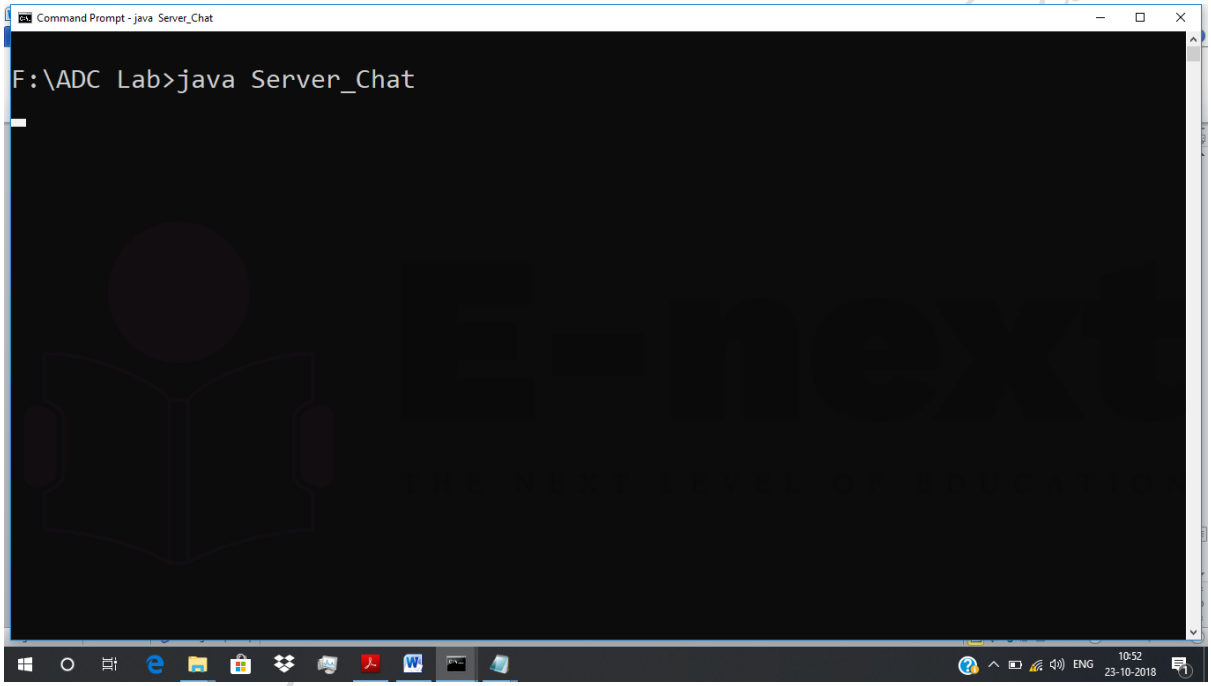
Client_Chat.java

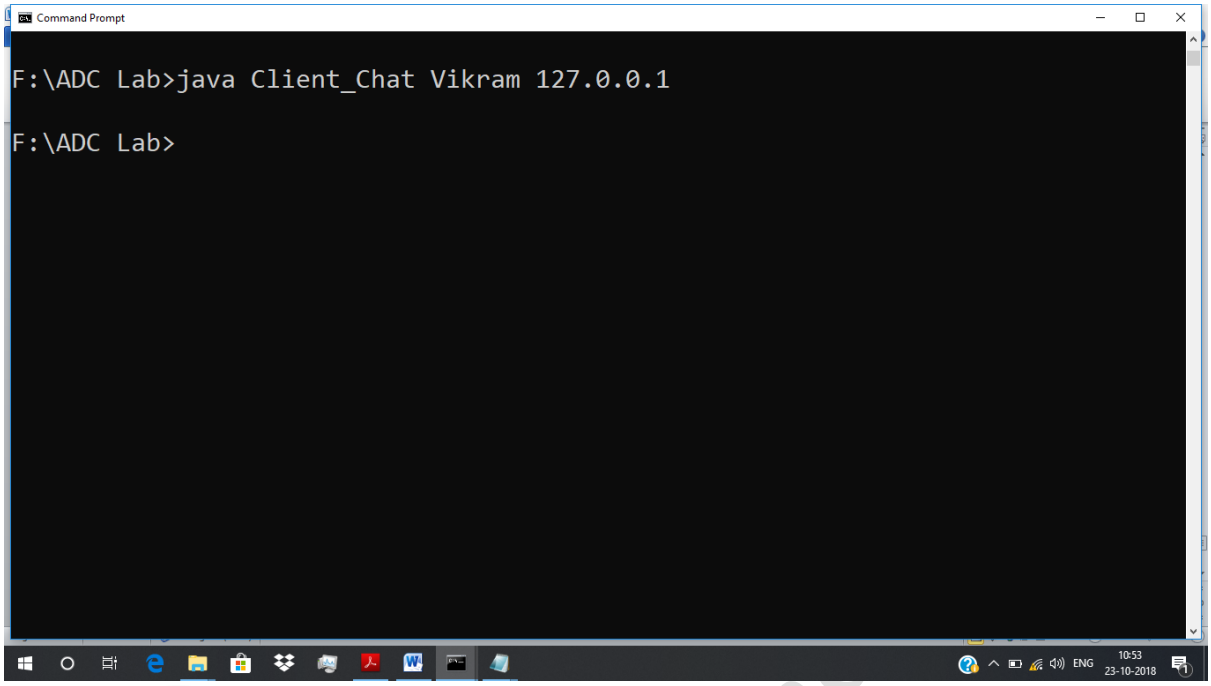
```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.*;
public class Client_Chat extends Frame implements Runnable
{
    TextArea ta;
     TextField tf;
    Button btnSend,btnClose;
    Socket soc;
    String sendTo,LoginName;
    DataOutputStream dout;
    DataInputStream din;
    Thread t;
   public Client_Chat(String LoginName,String chatwith) throws Exception
  {
     super(LoginName);
     this.LoginName = LoginName;
     sendTo = chatwith;
     ta = new TextArea(50,50);
     tf = new TextField(50);
     btnSend = new Button("Send");
```

```
        btnClose = new Button("Close");
        soc = new Socket(InetAddress.getLocalHost(),3000);
        din = new DataInputStream(soc.getInputStream());
        dout = new DataOutputStream(soc.getOutputStream());
        dout.writeUTF(LoginName);
        t  =new Thread(this);
        t.start();
     }
    void setup()
    {
        setSize(600,600);
        setLayout(new GridLayout(2,1));
        add(ta);
        Panel p = new Panel();
        p.add(tf);
        p.add(btnSend);
        p.add(btnClose);
        add(p);
        show();
     }
 public static void main(String args[]) throws Exception
 {
    Client_Chat client = new Client_Chat(args[0],args[1]);
    client.setup();
 }
public void run()
{
   while(true)
   {
     try
     {
       ta.append("\n"+sendTo+"Says:"+din.readUTF());
     }catch(Exception ex)
      {  ex.printStackTrace(); }
   }
}
public boolean action(Event e , Object o)
{
   if(e.arg.equals("Send"))
  {
     try
     {
      dout.writeUTF(sendTo+"  "+"DATA"+" "+tf.getText().toString());
      ta.append("\n"+LoginName+"says"+tf.getText().toString());
     }catch(Exception ex){ }
  }
    else if(e.arg.equals("Close"))
    {
       try
```

```
    {
    dout.writeUTF(LoginName+"LOGOUT");
    System.exit(1);
    }catch(Exception ex){ }
 }
return super.action(e,o);
}
}
```

Output:




Command Prompt
F:\ADC Lab>java Client_Chat Vikram 127.0.0.1
F:\ADC Lab>


Command Prompt - java Client_Chat Vikram 127.0.0.1
F:\ADC Lab>j
F:\ADC Lab>j
Vikram
VikramsaysVikram
127.0.0.1Says:I am offline
Vikram
Send
Close

Prog 6: GUI Based Chatting application between Server and Multiple Client

AppServer.java

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

public class AppServer extends Frame implements ActionListener,Runnable
{
 //Declarations
 Button b1;
 TextField tf;
 TextArea ta;
 ServerSocket ss;
 Socket s;
 PrintWriter pw;
 BufferedReader br;
 Thread th;
 public AppServer()
 {
  Frame f=new Frame("Server Side Chatting");//Frame for Server
  f.setLayout(new FlowLayout());//set layout
  f.setBackground(Color.orange);//set background color of the Frame
  b1=new Button("Send");//Send Button
  b1.setBackground(Color.pink);
  b1.addActionListener(this);//Add action listener to send button.
  tf=new TextField(15);
  ta=new TextArea(12,20);
  ta.setBackground(Color.cyan);
  f.addWindowListener(new W1());//add Window Listener to the Frame
  f.add(tf);//Add TextField to the frame
  f.add(b1);//Add send Button to the frame
  f.add(ta);//Add TextArea to the frame
  try{
   ss=new ServerSocket(12000);//Socket for server
   s=ss.accept();//accepts request from client
   System.out.println(s);
   //below line reads input from InputStreamReader
   br=new BufferedReader(new InputStreamReader(s.getInputStream()));
   //below line writes output to OutPutStream
   pw=new PrintWriter(s.getOutputStream(),true);
  }catch(Exception e)
  {
  }
  th=new Thread(this);//start a new thread
  th.setDaemon(true);//set the thread as demon
  th.start();
  setFont(new Font("Arial",Font.BOLD,20));
```

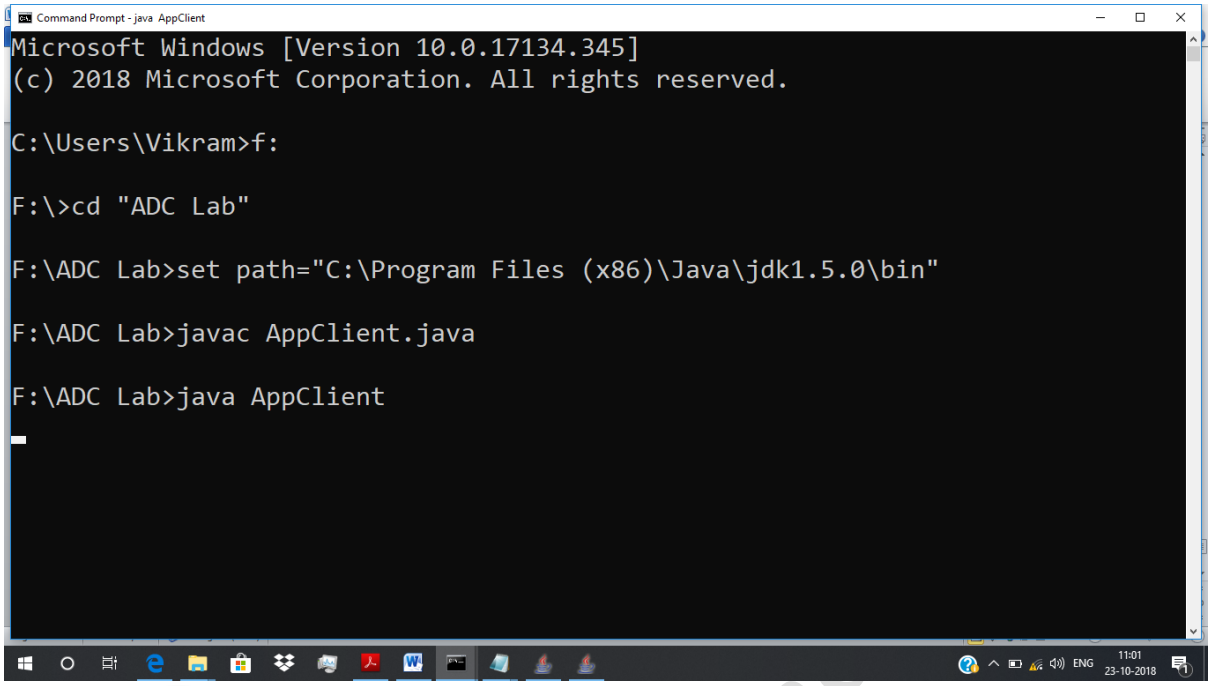
```
 f.setSize(200,200);//set the size
 f.setLocation(300,300);//set the location
 f.setVisible(true);
 f.validate();
}
//method required to close the Frame on clicking "X" icon.
private class W1 extends WindowAdapter
{
 public void windowClosing(WindowEvent we)
 {
  System.exit(0);
 }
}
//This method will called after clicking on Send button.
public void actionPerformed(ActionEvent ae)
{
 pw.println(tf.getText());//write the value of textfield into PrintWriter
 tf.setText("");//clean the textfield
}
//Thread running as a process in background
public void run()
{
 while(true)
 {
  try{
   String s=br.readLine();//reads the input from textfield
   ta.append(s+"\n");//Append to TextArea
  }catch(Exception e)
  {
  }
 }
}
//Main method
public static void main(String args[])
{
 //Instantiate AppServer class
 AppServer server = new AppServer();
}
}
```

AppClient.java

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

public class AppClient extends Frame implements ActionListener,Runnable
{
 Button b;
 TextField tf;
 TextArea ta;
 Socket s;
 PrintWriter pw;
 BufferedReader br;
 Thread th;
 public AppClient()
 {
  Frame f=new Frame("Client Side Chatting");
  f.setLayout(new FlowLayout());
  f.setBackground(Color.orange);
  b=new Button("Send");
  b.addActionListener(this);
  f.addWindowListener(new W1());
  tf=new TextField(15);
  ta=new TextArea(12,20);
  ta.setBackground(Color.cyan);
  f.add(tf);
  f.add(b);
  f.add(ta);
  try{
   s=new Socket(InetAddress.getLocalHost(),12000);
   br=new BufferedReader(new InputStreamReader(s.getInputStream()));
   pw=new PrintWriter(s.getOutputStream(),true);
  }
  catch(Exception e){}
  th=new Thread(this);
  th.setDaemon(true);
  th.start();
  setFont(new Font("Arial",Font.BOLD,20));
  f.setSize(200,200);
  f.setVisible(true);
  f.setLocation(100,300);
  f.validate();
 }
 private class W1 extends WindowAdapter
 {
  public void windowClosing(WindowEvent we)
  {
```

```
 System.exit(0);
 }
}
public void actionPerformed(ActionEvent ae)
{
 pw.println(tf.getText());
 tf.setText("");
}
public void run()
{
 while(true)
 {
  try{
   ta.append(br.readLine()+"\n");
  }catch(Exception e) {}
 }
}
public static void main(String args[])
{
 AppClient client = new AppClient();
}
}
```

Output :



Prog 7: Simple program for fetching current date and time from Server by Client

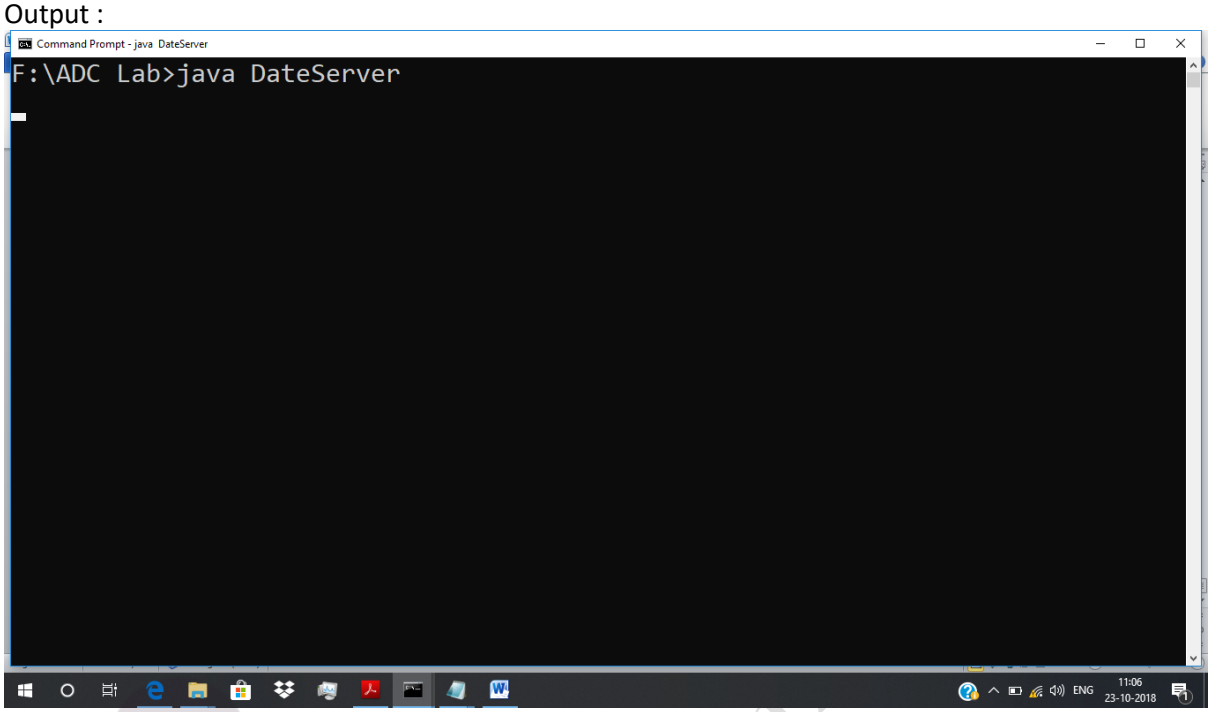
DateServer.java

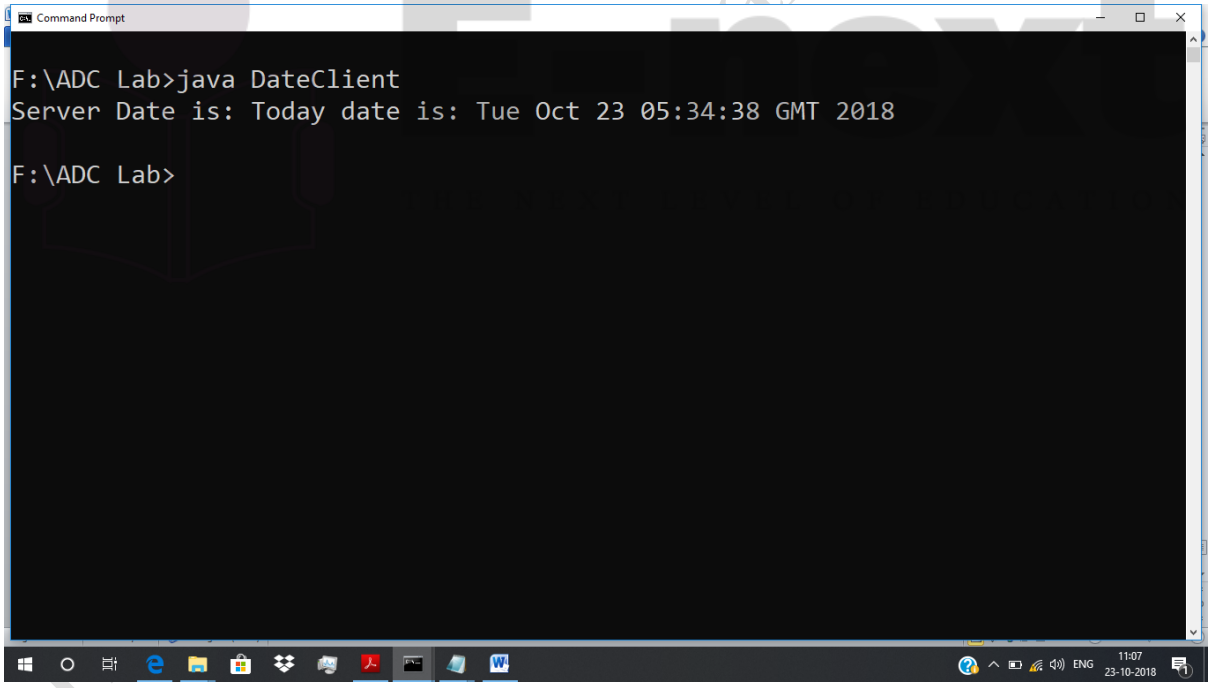
```
import java.net.*;
import java.io.*;
import java.util.*;
public class DateServer
{
 public static void main(String args[]) throws Exception
 {
   ServerSocket ss = new ServerSocket(3000);
   while(true)
   {
     Socket s = ss.accept();
    DataOutputStream out = new DataOutputStream(s.getOutputStream());
  out.writeBytes("Today date is: "+(new Date()).toString()+"\n");
  out.close();
  s.close();
 }
}
}
```

DateClient.java

```
import java.net.*;
import java.io.*;
public class DateClient
{
 public static void main(String args[]) throws Exception
{
  Socket soc = new Socket(InetAddress.getLocalHost(),3000);
  BufferedReader br = new BufferedReader(new InputStreamReader(soc.getInputStream()));
System.out.println("Server Date is: "+br.readLine());
}
}
```

Output :


Command Prompt - java DateServer
F:\ADC Lab>java DateServer


Command Prompt
F:\ADC Lab>java DateClient
Server Date is: Today date is: Tue Oct 23 05:34:38 GMT 2018
F:\ADC Lab>

Prog. 8 : Online Gossiping chatting between Client and Server

GossipServer.java

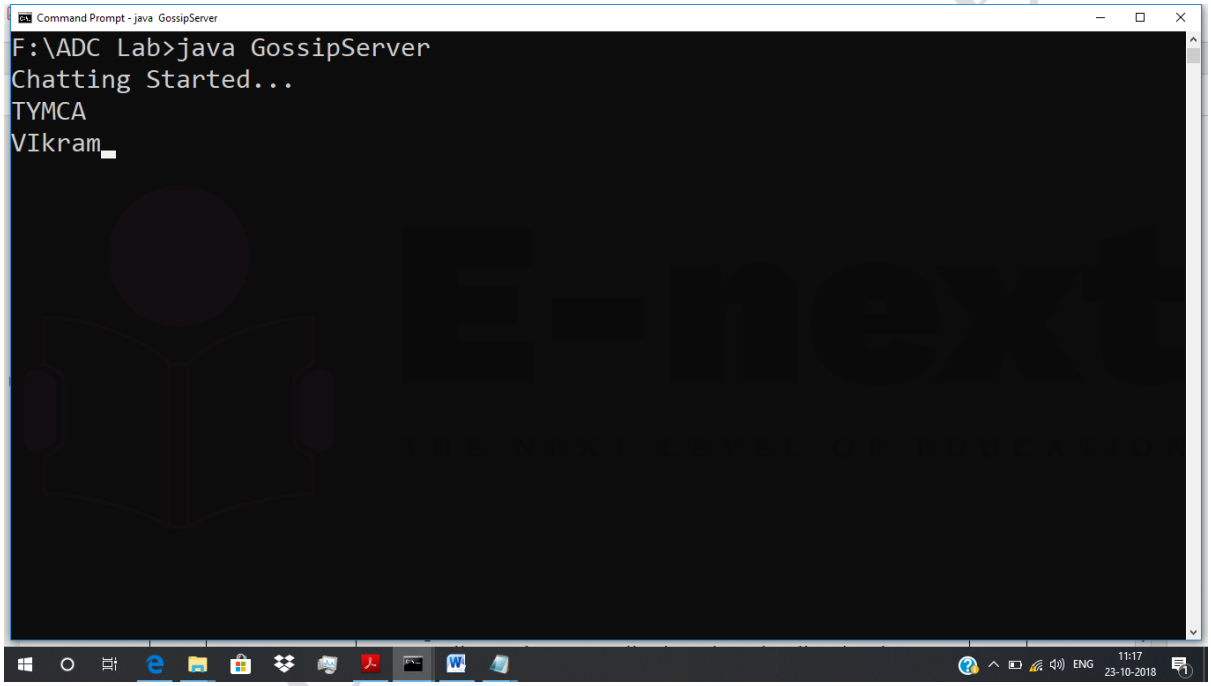
```
import java.net.*;
import java.io.*;
public class GossipServer
{
  public static void main(String args[]) throws Exception
{
  ServerSocket sersock = new ServerSocket(3000);
  System.out.println("Chatting Started...");
  Socket sock = sersock.accept();
 BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
OutputStream ostream = sock.getOutputStream();
PrintWriter pwrite = new PrintWriter(ostream,true);
InputStream istream = sock.getInputStream();
BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
String receiveMessage,sendMessage;
while(true)
{
if((receiveMessage = receiveRead.readLine())!=null)
{
 System.out.println(receiveMessage);
}
sendMessage = keyRead.readLine();
pwrite.println(sendMessage);
pwrite.flush();
}
}
}
```

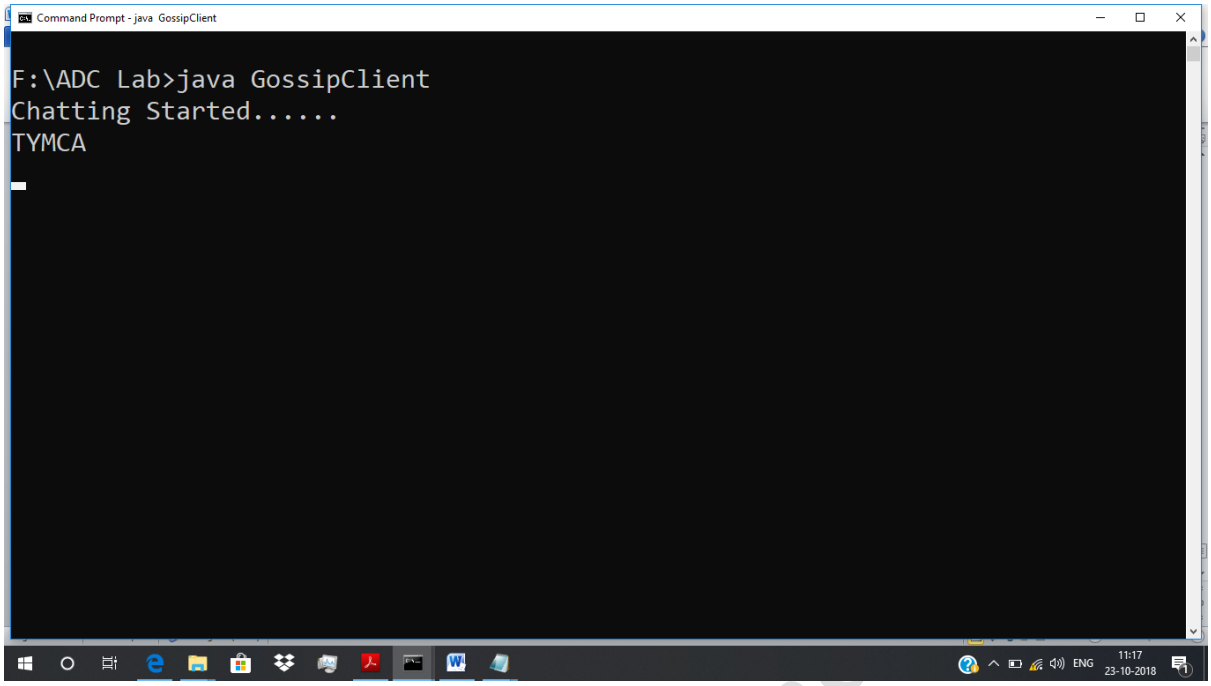
GossipClient.java

```
import java.io.*;
import java.net.*;
public class GossipClient
{
 public static void main(String args[]) throws Exception
 {
   Socket sock = new Socket(InetAddress.getLocalHost(),3000);
  BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
OutputStream ostream = sock.getOutputStream();
PrintWriter pwrite = new PrintWriter(ostream,true);
InputStream istream = sock.getInputStream();
BufferedReader read = new BufferedReader(new InputStreamReader(istream));
System.out.println("Chatting Started.....");
String receiveMessage,sendMessage;
while(true)
```

```
{
  sendMessage = keyRead.readLine();
   pwrite.println(sendMessage);
   pwrite.flush();
if((receiveMessage = read.readLine())!=null)
{
  System.out.println(receiveMessage);
}
}
}
}
```

Output :





Prog. 9 : Remote Procedure Call (RPC) program to implement server calculator containing basic mathematical methods.

Code :

ServerRPC.java

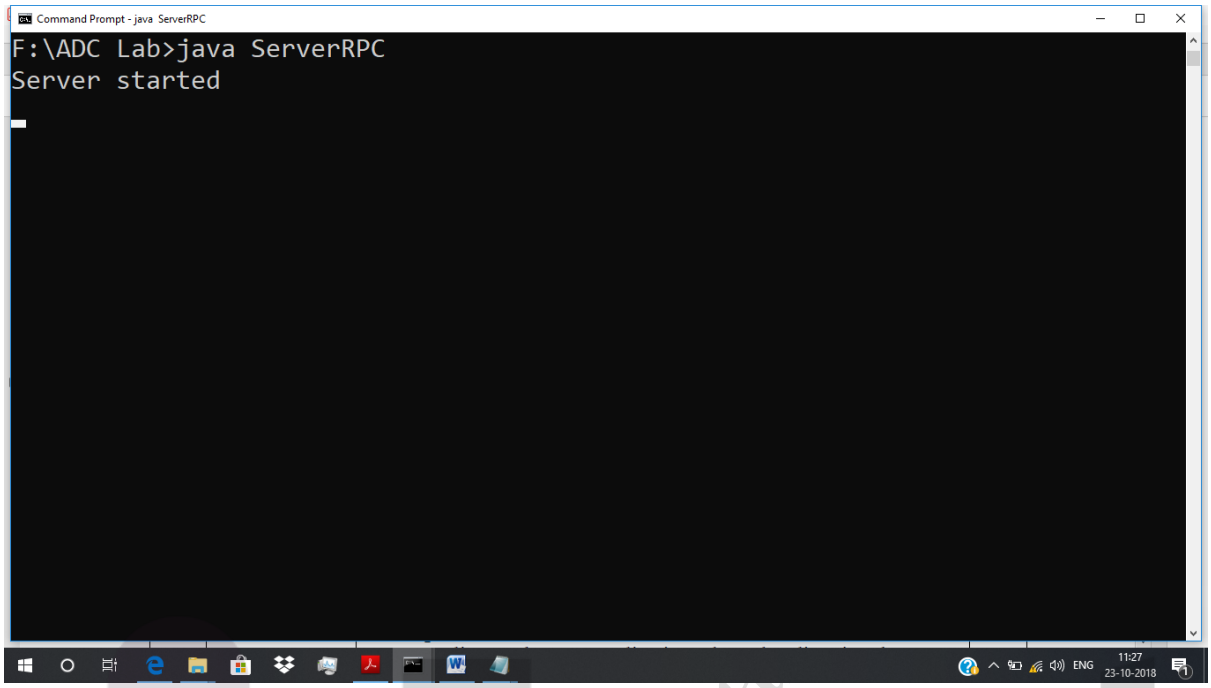
```
import java.net.*;
import java.io.*;

public class ServerRPC
{
  public static void main(String args [] ) throws Exception
{
  ServerSocket sersock = new ServerSocket(3000);
System.out.println("Server started");
        Socket  csock = sersock.accept();
OutputStream ostream = csock.getOutputStream();
PrintWriter out = new PrintWriter(ostream,true);

InputStream istream = csock.getInputStream();
BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
int a,b,c;
String receiveMessage,sendMessage,function;
while(true)
{
 function = receiveRead.readLine();
 if(function!=null)
  System.out.println("Operation is: "+function);
  a = Integer.parseInt(receiveRead.readLine());
  System.out.println("First Number: "+a);
  b = Integer.parseInt(receiveRead.readLine());
  System.out.println("Second Number: "+b);
  if(function.compareTo("add")==0)
  {
      c = a+b;
      out.println("Addition is: "+c);
  }
 if(function.compareTo("sub")==0)
 {
    c = a-b;
    out.println("Substraction is: "+c);
 }
 if(function.compareTo("mul")==0)
  {
    c = a*b;
   out.println("Multiplication is: "+c);
  }
if(function.compareTo("div")==0)
```

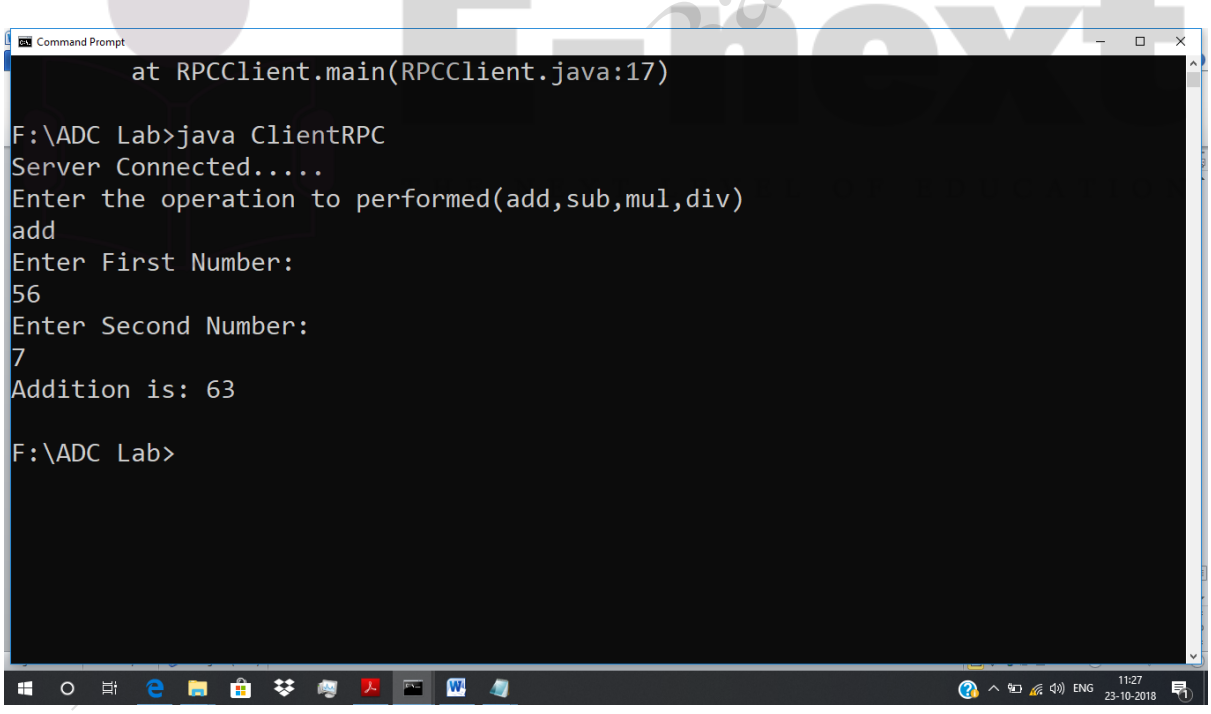
```
 {
   c  = a / b;
 out.println("Division is: "+c);
 }
  out.flush();

}

}
}
```

ClientRPC.java

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

public class ClientRPC
{
 public static void main(String args[]) throws Exception
{
  Socket sock = new Socket(InetAddress.getLocalHost(),3000);
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
OutputStream ostream = sock.getOutputStream();
PrintWriter out = new PrintWriter(ostream,true);

InputStream istream = sock.getInputStream();
BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
System.out.println("Server Connected.....");
String no1,no2,temp,receiveMessage,sendMessage;
System.out.println("Enter the operation to performed(add,sub,mul,div)");
 temp = br.readLine();
 sendMessage = temp.toLowerCase();
 out.println(sendMessage);
System.out.println("Enter First Number: ");
 no1 = br.readLine();
 out.println(no1);
System.out.println("Enter Second Number: ");
 no2 = br.readLine();
out.println(no2);
out.flush();
if((receiveMessage=receiveRead.readLine())!=null)
 System.out.println(receiveMessage);
}
}
```

Output :


Command Prompt - java ServerRPC
F:\ADC Lab>java ServerRPC
Server started


Command Prompt
at RPCClient.main(RPCClient.java:17)
F:\ADC Lab>java ClientRPC
Server Connected.....
Enter the operation to performed(add,sub,mul,div)
add
Enter First Number:
56
Enter Second Number:
7
Addition is: 63
F:\ADC Lab>

Prog 10: Simple Chatting program for Server and Client

ChatServer.java

```
import java.io.*;
import java.net.*;
public class ChatServer extends Thread
{
  ServerSocket ss;
  Socket soc;
  BufferedReader br,br1;
  PrintWriter out;
  String str;
  public ChatServer()
  {
   try
   {
   ss = new ServerSocket(9999);
   soc = ss.accept();
   br = new BufferedReader(new InputStreamReader(soc.getInputStream()));
 System.out.println("Chat Server Started.....");
     start();
    new InnerServer();
  }catch(Exception e){ }
 } // close constructor
  public void run()
{
  try
  {
    while(true)
   {
   str = br.readLine();
   System.out.println("Client Message is: "+str);
   }
 }catch(Exception e){ }
}

class InnerServer
{
  String str1;
  InnerServer()
  {
     try
     {
    br1 = new BufferedReader(new InputStreamReader(System.in));
  out = new PrintWriter(soc.getOutputStream(), true);
  while(true)
 {
   str1 = br1.readLine();
```

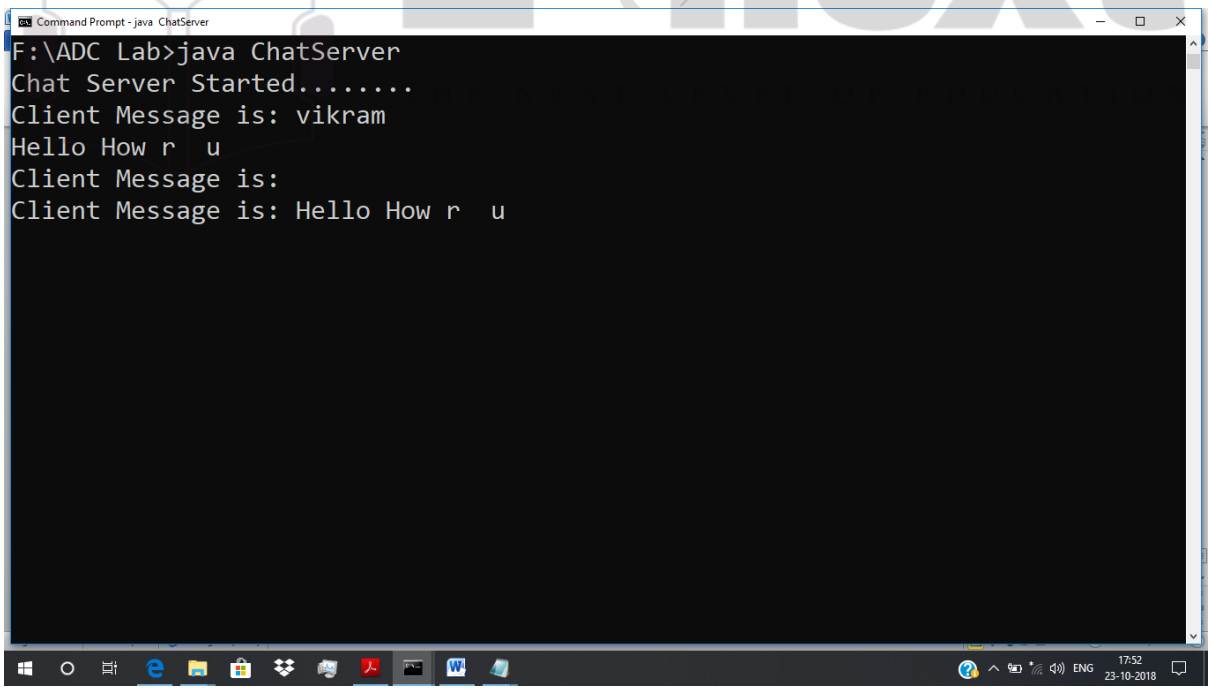
```
  out.println(str1);
 }
 }catch(Exception e){}
}
}

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

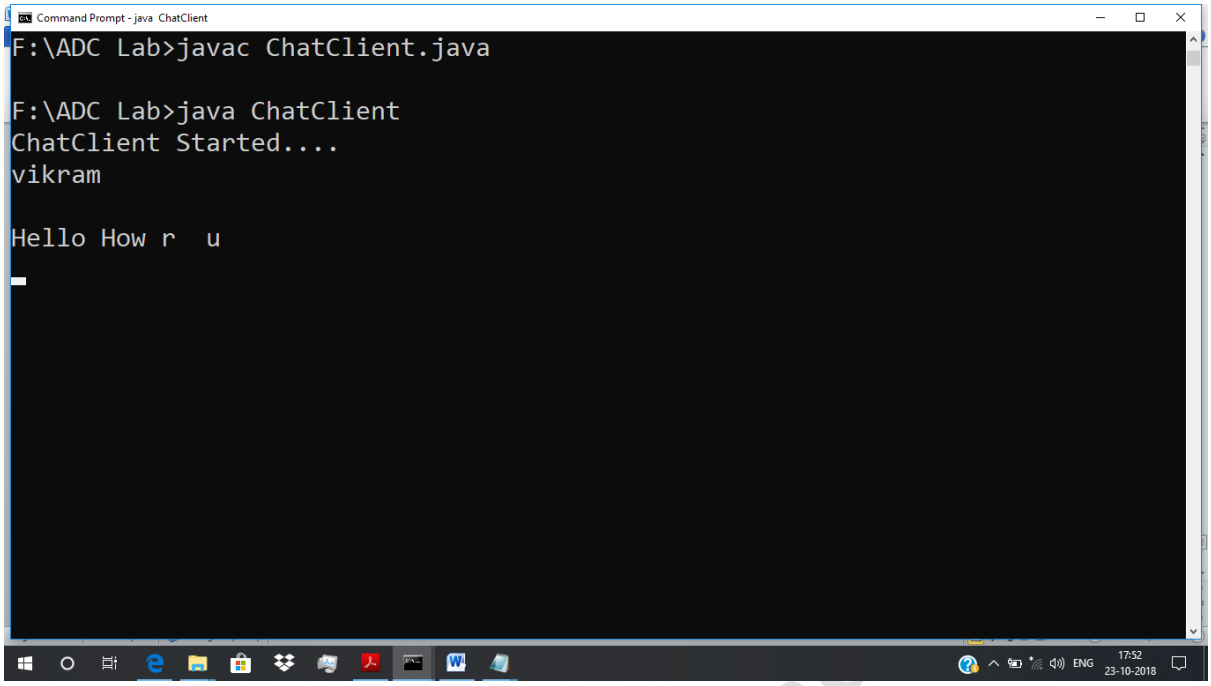
ChatClient.java

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

public class ChatClient
{
 Socket soc;
 BufferedReader br,br1;
 PrintWriter out;
 String str;
 public ChatClient()
{
 try
 {
 soc = new Socket(InetAddress.getLocalHost(),9999);
 br = new BufferedReader(new InputStreamReader(System.in));
 out = new PrintWriter(soc.getOutputStream(),true);
 System.out.println("ChatClient Started....");
 while(true)
 {
  str = br.readLine();
  out.println(str);

  new InnerClient();
 }
}catch(Exception e){ }
}
class InnerClient extends Thread
{
  String str1;
  InnerClient()
  {
   try
   {
   br1 = new BufferedReader(new InputStreamReader(soc.getInputStream()));
  start();
```

```
   }catch(Exception e) { }
 }
}
 public void run()
 {
  try
  {
   while(true)
  {
     str = br.readLine();
    System.out.println("Server Says: "+str);
   }
 }catch(Exception e)
{ }
}

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

Output :





Prog. 11. Implement Date and Time Server and display on Client.

RPCServer1.java

```
import java.util.*;
import java.net.*;
import java.util.*;
import java.text.SimpleDateFormat;
public class RPCServer1
 {
        DatagramSocket ds;
        DatagramPacket dp;
        String str,methodName,result;
        int val1,val2;

        RPCServer1()
        {
                try
                {
                ds=new DatagramSocket(1200);
                byte b[]=new byte[4096];

                while(true)
                {
                dp=new DatagramPacket(b,b.length);
                ds.receive(dp);
                str=new String(dp.getData(),0,dp.getLength());
                        if(str.equalsIgnoreCase("q"))
                 {
                        System.exit(1);
                  }
                else
                  {
        StringTokenizer st = new StringTokenizer(str," ");
                        int i=0;

                        while(st.hasMoreTokens())
                        {
                                String token=st.nextToken();
                               methodName=token;
                               }
                        }
                Calendar c = Calendar.getInstance();
                SimpleDateFormat dateFormat =
new SimpleDateFormat("dd/MM/yyyy");
                Date d = c.getTime();
                InetAddress ia = InetAddress.getLocalHost();
                if(methodName.equalsIgnoreCase("date"))
                 {
                        result=""+ dateFormat.format(d);
```

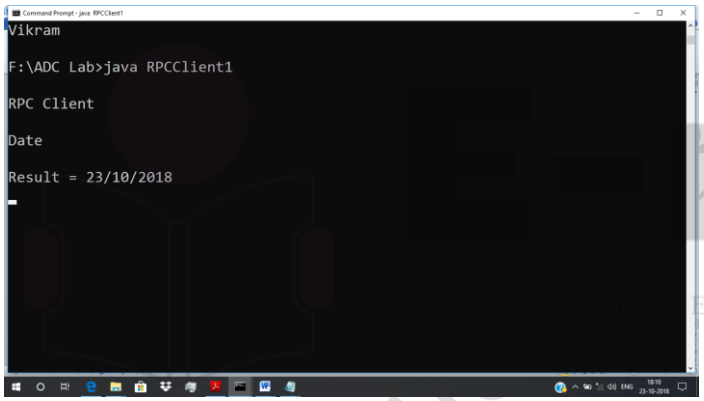
```
                }
                else if(methodName.equalsIgnoreCase("time"))
                {
                        result= "" + d.getHours() +" : " +d.getMinutes() +
" : " +d.getSeconds();
                }
                byte b1[]=result.getBytes();
                DatagramSocket ds1 = new DatagramSocket();
DatagramPacket dp1 = new   DatagramPacket
 (b1,b1.length,InetAddress.getLocalHost(), 1300);
                System.out.println("result : "+result+"\n");
                ds1.send(dp1);
                }}
        catch (Exception e)
        {
                        e.printStackTrace();
        }}
        public static void main(String[] args)
        {
                new RPCServer1();
        }
}
```

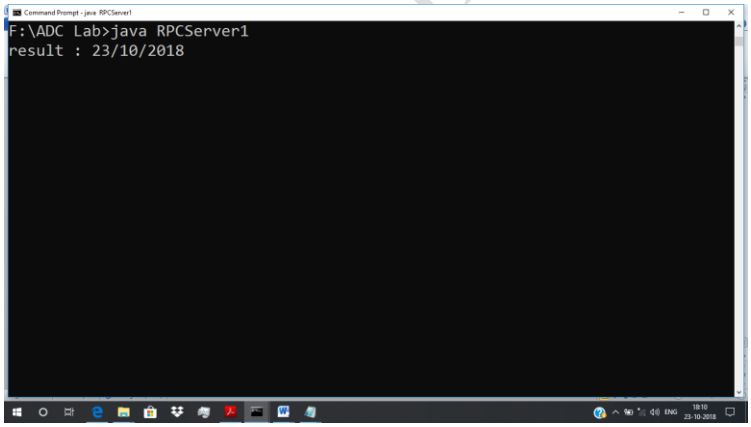
RPCClient1.java

```
import java.io.*;
import java.net.*;
public class RPCClient1
{
        public RPCClient1()
        {
                try
                {
        InetAddress ia = InetAddress.getLocalHost();
        DatagramSocket ds = new DatagramSocket();
        byte b1[]=new byte[50];
        DatagramSocket ds1 = new DatagramSocket(1300);
        System.out.println("\nRPC Client\n");

                        while (true)
                        {
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                        String str = br.readLine();
                        byte b[] = str.getBytes();
                        DatagramPacket dp = new DatagramPacket(b,b.length,ia,1200);
                        ds.send(dp);

        dp = new DatagramPacket(b1,b1.length);
        ds1.receive(dp);
```

```
		String s = new String(dp.getData(),0,dp.getLength());
		System.out.println("\nResult = " + s );
				}
			}
			catch (Exception e)
			{
				e.printStackTrace();
			}
		}
public static void main(String[] args)
		{
			new RPCClient1();
		}
}
```

Output :





Prog. 12 : Remote Method Invocation (RMI) program for Distributed Computing in java. Addition of two numbers using Client – Server using RMI.

Code:

(I) AddServerIntf.java

```
import java.rmi.*;
public interface AddServerIntf extends Remote
{
double add(double d1 , double d2) throws RemoteException;
}
```

(II) AddServer.java

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

public class AddServer
{
public static void main(String args[])
{
try
{
AddServerImpl ads = new AddServerImpl();
Naming.rebind("AddServer", ads);
}
catch(Exception e)
{ System.out.println(e); }
}
}
```

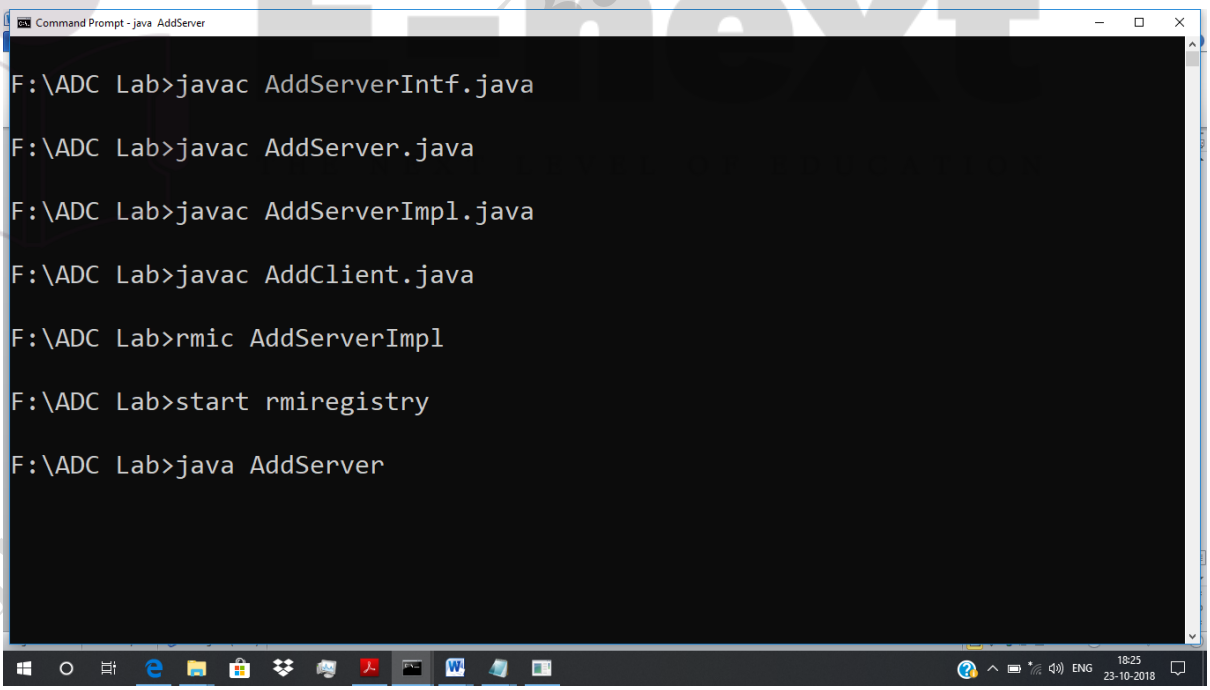
(III) AddServerImpl.java

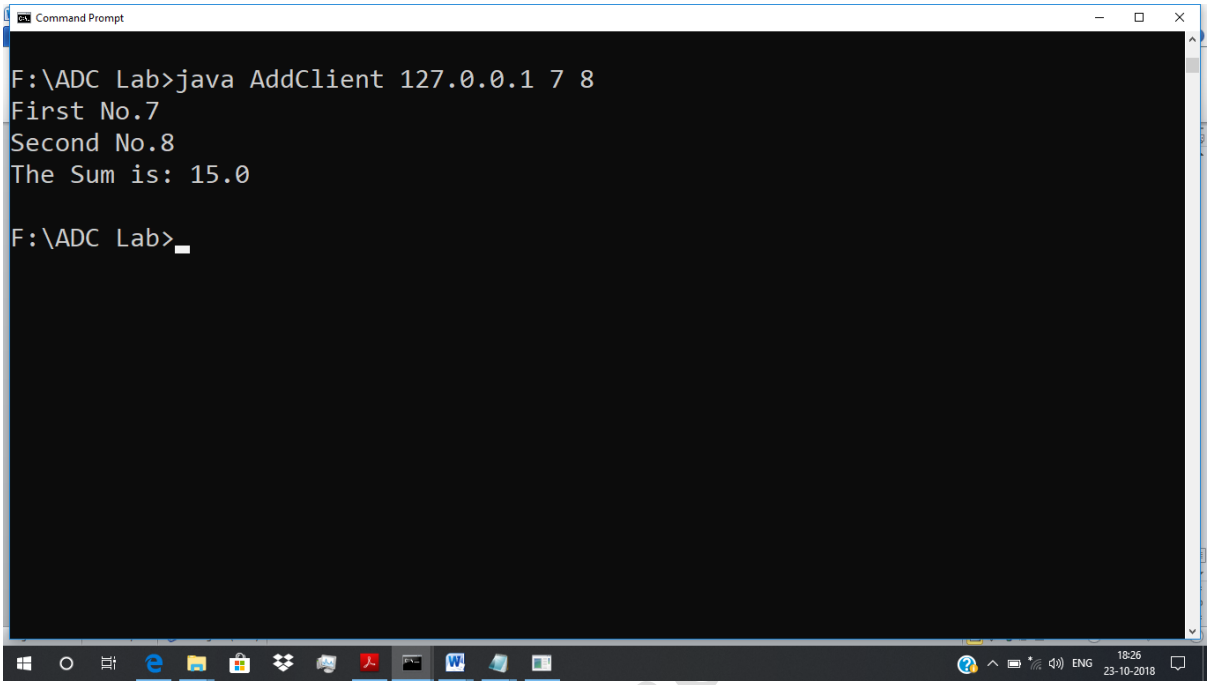
```
import java.rmi.*;
import java.rmi.server.*;
public class AddServerImpl extends UnicastRemoteObject
implements AddServerIntf
{
public AddServerImpl() throws RemoteException
{ }
public double add(double d1 , double d2) throws RemoteException
{
return d1+d2;
}
}
```

(IV) AddClient.java

```
import java.rmi.*;
public class AddClient
{
public static void main(String args[])
{
try
{
String URL = "rmi://"+args[0]+"/AddServer";
AddServerIntf addserverIntf = (AddServerIntf)Naming.lookup(URL);
System.out.println("First No."+args[1]);
System.out.println("Second No."+args[2]);
double d1 = Double.valueOf(args[1]).doubleValue();
double d2 = Double.valueOf(args[2]).doubleValue();
System.out.println("The Sum is: "+addserverIntf.add(d1,d2));
}catch(Exception e){ }
}
}
```

Output :





Prog. 13 : Retrieve Time and Date from Server to Client using RMI.

Code :

(I) DateIntf.java

```
// Retrive Date from Server using RMI
import java.rmi.*;

public interface DateIntf extends Remote
{
   public String Today_Date() throws RemoteException;
}
```

(II) DateImpl.java

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

public class DateImpl extends UnicastRemoteObject implements DateIntf
{
    public DateImpl() throws RemoteException
    {    }
   public String Today_Date() throws RemoteException
   {
       return "Today Date and Time is"+new java.util.Date()+" ";
   }
}
```

(III) DateRMIServer.java

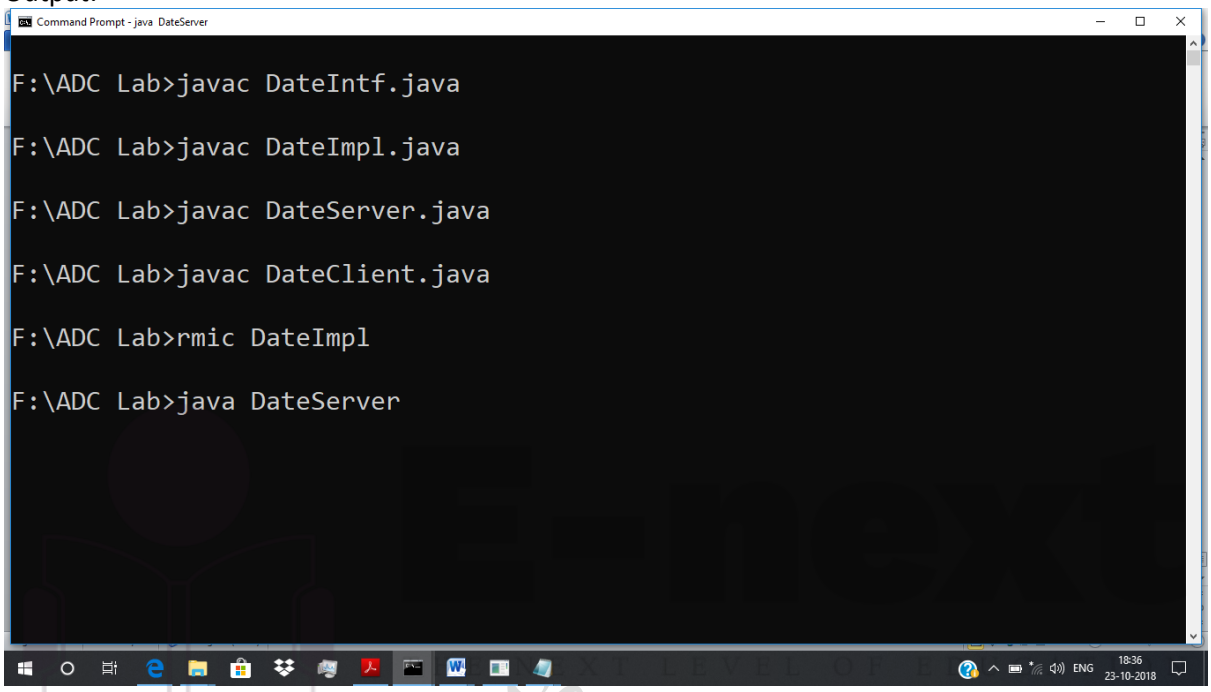
```
import java.net.*;
import java.rmi.*;
public class DateRMIServer
{
  public static void main(String args[]) throws Exception
{
   DateImpl di = new DateImpl();
   Naming.rebind("DateDemo",di);
 }
}
```

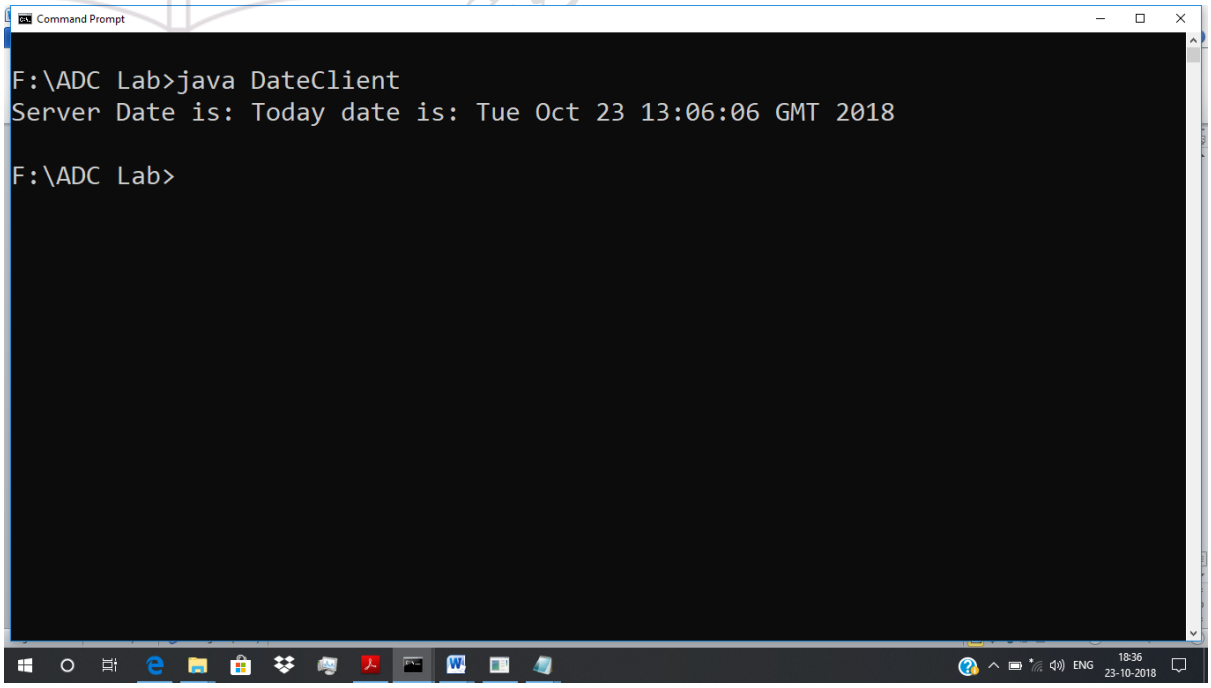
(IV) DateRMIClient.java

```
import java.net.*;
import java.rmi.*;
public class DateRMIClient
{
  public static void main(String args[]) throws Exception
```

```
  {
    String url = "rmi://"+args[0]+"/DateDemo";
    DateIntf dateintf = (DateIntf)Naming.lookup(url);
   System.out.println(" "+dateintf.Today_Date());
  }
}
```

Output:





Prog. 14. Equation Solver using RMI.

Client provides linear equation to server and Server solve the linear equation expression and return the result to client.

(I) LinearIntf.java

```
import java.rmi.*;

public interface LinearIntf extends Remote
{
  public double equation(double a , double b) throws RemoteException;
}
```

(II) LinearImpl.java

```
import java.rmi.*;
import java.net.*;
import java.rmi.server.*;

public class LinearImpl extends UnicastRemoteObject implements LinearIntf
{
   public LinearImpl() throws RemoteException
   { }
 public double equation(double a , double b) throws RemoteException
{
    return ((a*a)-(2*a*b)+(b*b));

}
}
```

(III) LinearServer.java

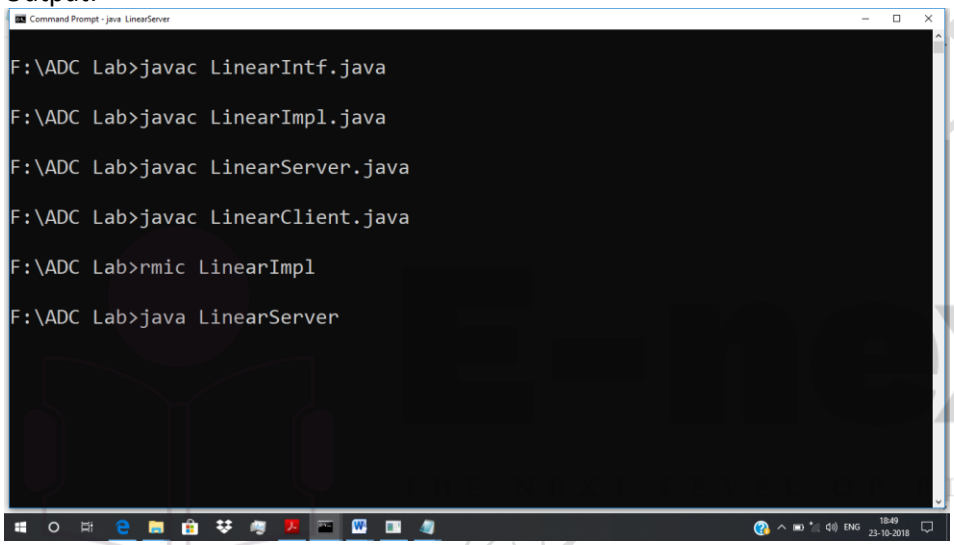
```
import java.net.*;
import java.rmi.*;

public class LinearServer
{
 public static void main(String args[]) throws Exception
{
   LinearImpl li = new LinearImpl();
   Naming.rebind("lineEq",li);
  }
}
```

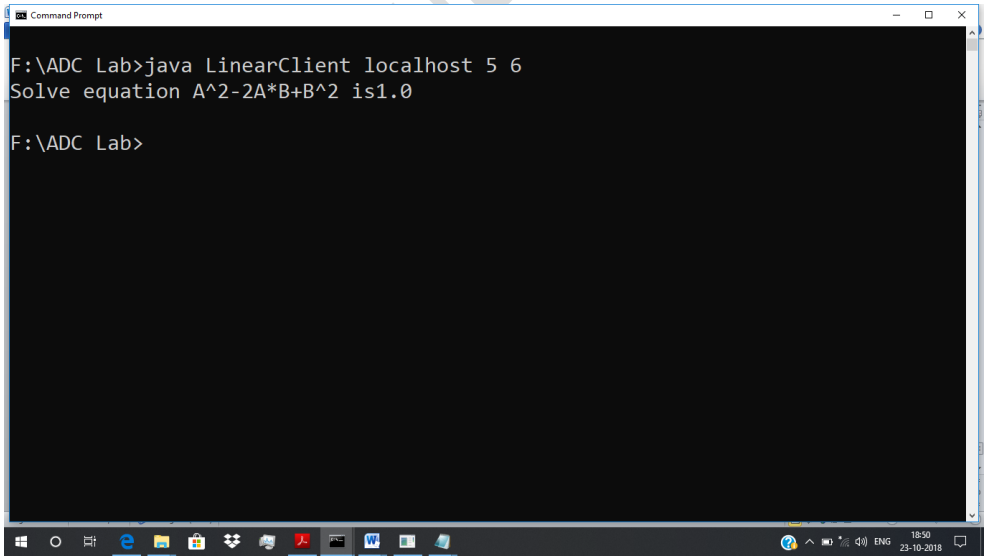
(IV) LinearClient.java

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

```
public class LinearClient
{
   public static void main(String args[]) throws Exception
{
  String url = "rmi://"+args[0]+"/lineEq";
  LinearIntf linearintf = (LinearIntf)Naming.lookup(url);
  double d1 = Double.valueOf(args[1]).doubleValue();
  double d2 = Double.valueOf(args[2]).doubleValue();
  System.out.println("Solve equation A^2-2A*B+B^2 is"+linearintf.equation(d1,d2));
 }
}
```

Output:





Prog. 14 . Design a simple GUI for standard Calculator. And Operation are perform on Server and retrieve result by client from Server using Remote Method Invocation (RMI).

Code:

(i) ICalculator.java

```
import java.rmi.*;
public interface ICalculator extends Remote
{
double add(double x,double y) throws RemoteException;
double sub(double x,double y) throws RemoteException;
double mul(double x,double y) throws RemoteException;
double div(double x,double y) throws RemoteException;
}
```

(ii) CalculatorImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
public class CalculatorImpl extends UnicastRemoteObject implements ICalculator
{
public CalculatorImpl() throws RemoteException
{
}
public double add(double x,double y) throws RemoteException
{
return(x+y);
}
public double sub(double x,double y) throws RemoteException
{
return(x-y);
}
public double mul(double x,double y) throws RemoteException
{return(x*y);
}
public double div(double x,double y) throws RemoteException
{
return(x/y);
}
}
```

(iii) CalculatorServer.java

```
import java.net.*;
import java.rmi.*;
public class CalculatorServer
{
public static void main(String[] args)
```

```
{
try
{
CalculatorImpl ci=new CalculatorImpl();
Naming.rebind("CalculatorServer1",ci);
System.out.println("Calculator Server is Ready");
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
```

(iv) CalculatorClient.java

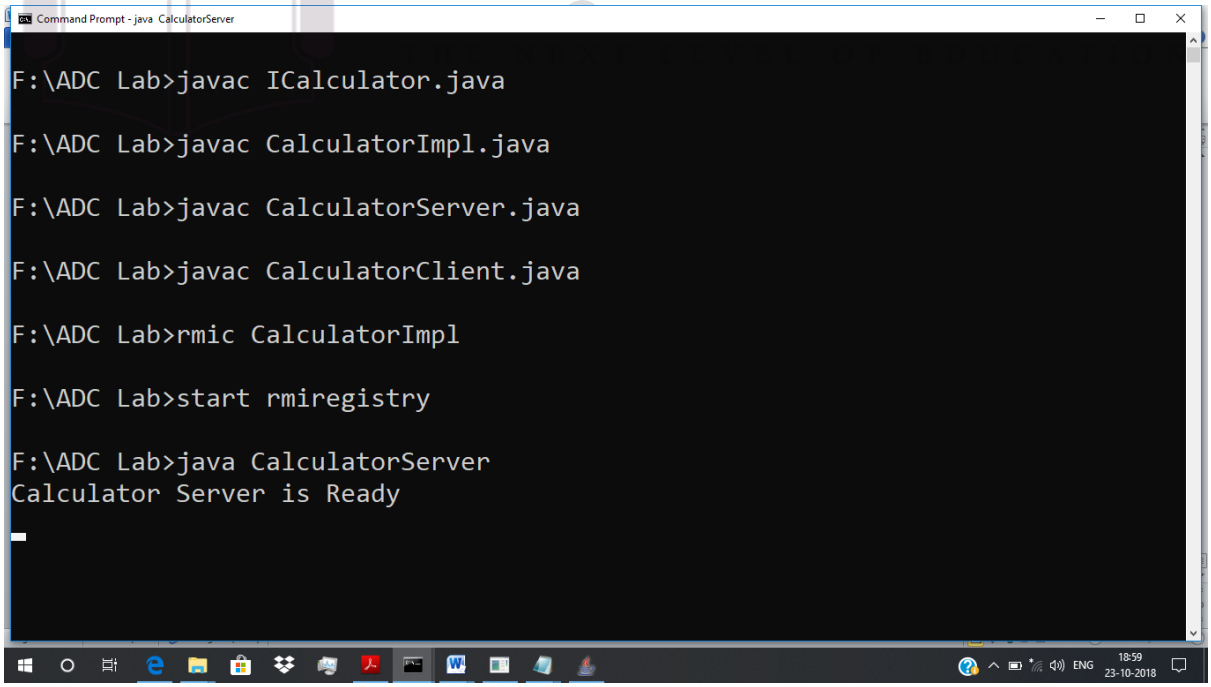
```
import java.rmi.*;
import java.net.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CalculatorClient extends JFrame implements ActionListener
{
String strNum1="",strNum2="",strRes="",op="";
double x,y,result;
boolean flag,dotFlag,resFlag;
ICalculator intf;
GridBagConstraints gbc=new GridBagConstraints();
JTextField txt1=new JTextField(20);
JButton btn[]=new JButton[17];
int i,j,k;
Container con;
public CalculatorClient()
{
con=this.getContentPane();
con.setLayout(new GridBagLayout());
gbc.weightx=1.0;
gbc.weighty=1.0;
btn[0]=new JButton("C");
btn[1]=new JButton("1");
btn[2]=new JButton("2");
btn[3]=new JButton("3");
btn[4]=new JButton("+");
btn[5]=new JButton("4");
btn[6]=new JButton("5");
btn[7]=new JButton("6");
btn[8]=new JButton("-");
btn[9]=new JButton("7");
```

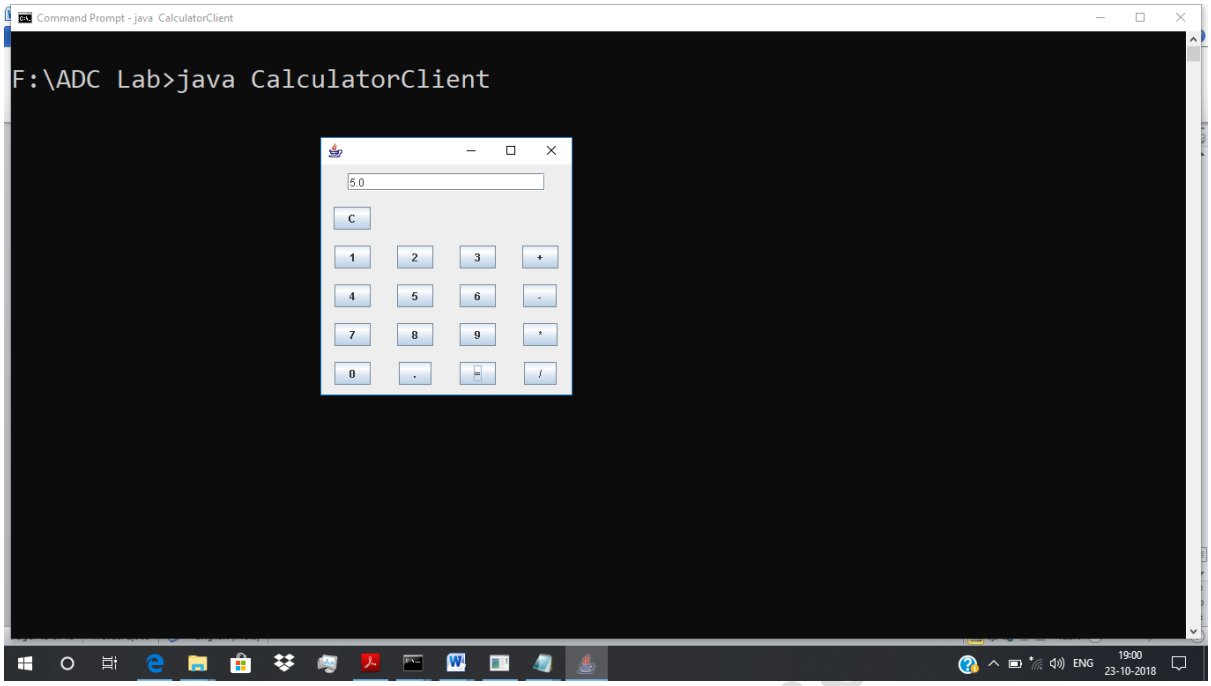
```
btn[10]=new JButton("8");
btn[11]=new JButton("9");
btn[12]=new JButton("*");
btn[13]=new JButton("0");
btn[14]=new JButton(".");
btn[15]=new JButton("=");
btn[16]=new JButton("/");
gbc.gridx=0;
gbc.gridy=0;
gbc.gridwidth=4;
con.add(txt1,gbc);
gbc.gridwidth=1;
gbc.gridx=0;
gbc.gridy=1;
con.add(btn[0],gbc);
btn[0].addActionListener(this);
i=1;
for(k=2;k<=5;k++)
{
for(j=0;j<=3;j++)
{
gbc.gridx=j;
gbc.gridy=k;
con.add(btn[i],gbc);
btn[i].addActionListener(this);
i++;
}
}
setSize(300,300);
setVisible(true);
}
public void actionPerformed(ActionEvent ae)
{
try
{
String url="rmi://127.0.0.1/CalculatorServer1";
intf = (ICalculator)Naming.lookup(url);
}
catch(Exception e)
{
e.printStackTrace();
}
String cmd=ae.getActionCommand();
if(cmd.equals("C"))
{
txt1.setText("");
strNum1=strNum2=strRes="";
x=y=result=0;
flag=true;
```

```
dotFlag=false;
resFlag=false;
}
else if(cmd.equals("+") || cmd.equals("-") || cmd.equals("*")
||cmd.equals("/"))
{
if(flag)
{
strNum1=txt1.getText();
strNum2="";
x=Double.parseDouble(strNum1);
flag=false;
dotFlag=false;
resFlag=false;
}
txt1.setText("");
op=cmd;
}
else if(cmd.equals("="))
{
strNum2=txt1.getText();
y=Double.parseDouble(strNum2);
try
{
if(op.equals("+"))
{
result=intf.add(x,y);
}
if(op.equals("-"))
{
result=intf.sub(x,y);
}
if(op.equals("*"))
{
result=intf.mul(x,y);
}
if(op.equals("/"))
{
result=intf.div(x,y);
}
}
catch(Exception e)
{
e.printStackTrace();
}
txt1.setText(Double.toString(result));
dotFlag=true;
resFlag=true;
flag=true;
```

```
}
else if(cmd.equals("."))
{
if(!dotFlag)
{
txt1.setText(txt1.getText()+cmd);
dotFlag=true;
}
}
else
{
if(!resFlag)
{
txt1.setText(txt1.getText()+cmd);
}
}
}
public static void main(String[] args)
{
CalculatorClient cc=new CalculatorClient();
}
}
```

Output :





Prog 15 : Solving multiple linear equation using RMI

Code:

(i) intfEqSolve.java

```
import java.rmi.*;
public interface intfEqSolve extends Remote
{
public int solveEq1(int a,int b)throws RemoteException;
public int solveEq2(int a,int b)throws RemoteException;
public int solveEq3(int a,int b)throws RemoteException;
public int solveEq4(int a,int b)throws RemoteException;
}
```

(ii) implEqSolve.java

```
import java.rmi.*;
import java.rmi.server.*;
public class implEqSolve extends UnicastRemoteObject implements
intfEqSolve
{
public implEqSolve()throws RemoteException{}
public int solveEq1(int a,int b)throws RemoteException
{
int ans=(a*a)-(2*a*b)+(b*b);
return ans;
}
public int solveEq2(int a,int b)throws RemoteException
{
int ans=(a*a)+(2*a*b)+(b*b);
return ans;
}
public int solveEq3(int a,int b)throws RemoteException
{
int ans=(a*a*a)-(3*a*a*b)+(3*a*b*b)-(b*b*b);
return ans;
}
public int solveEq4(int a,int b)throws RemoteException
{
 int ans=(a*a*a)+(3*a*a*b)+(3*a*b*b)+(b*b*b);
return ans;
}
}
```

(iii) serverEqSolve.java

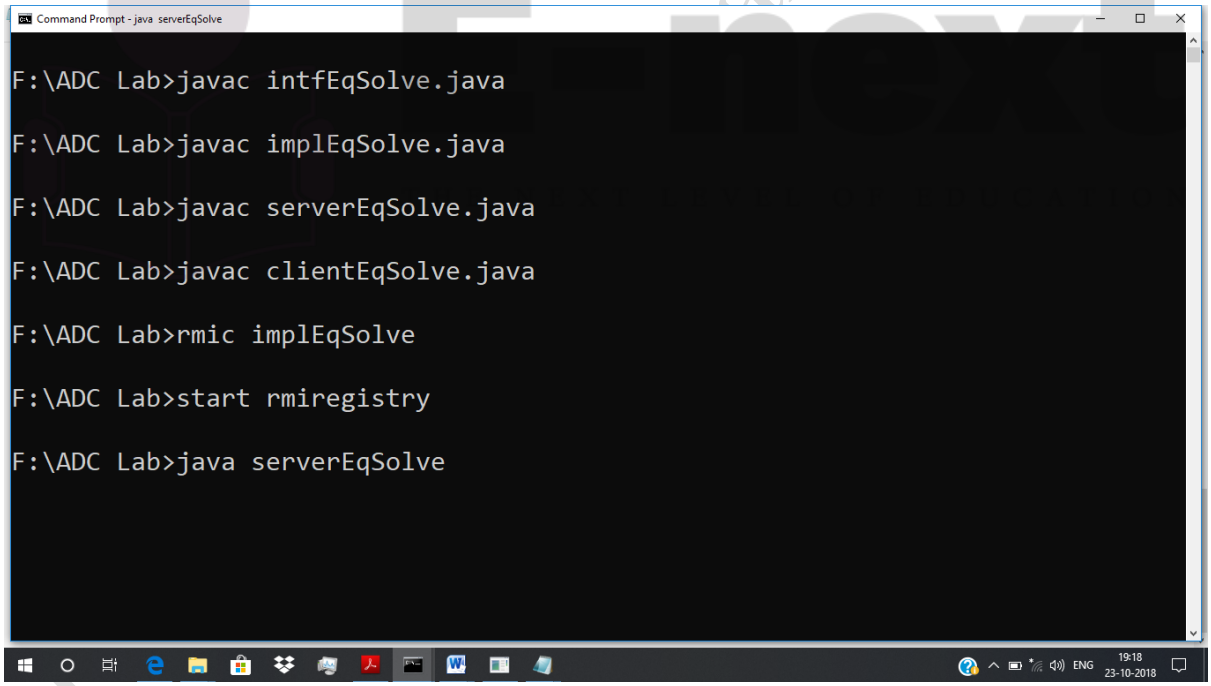
```
import java.io.*;
 import java.net.*;
import java.rmi.*;
public class serverEqSolve {
public static void main(String[]args)
{
try
{
implEqSolve obj=new implEqSolve();
Naming.rebind("hello",obj);
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

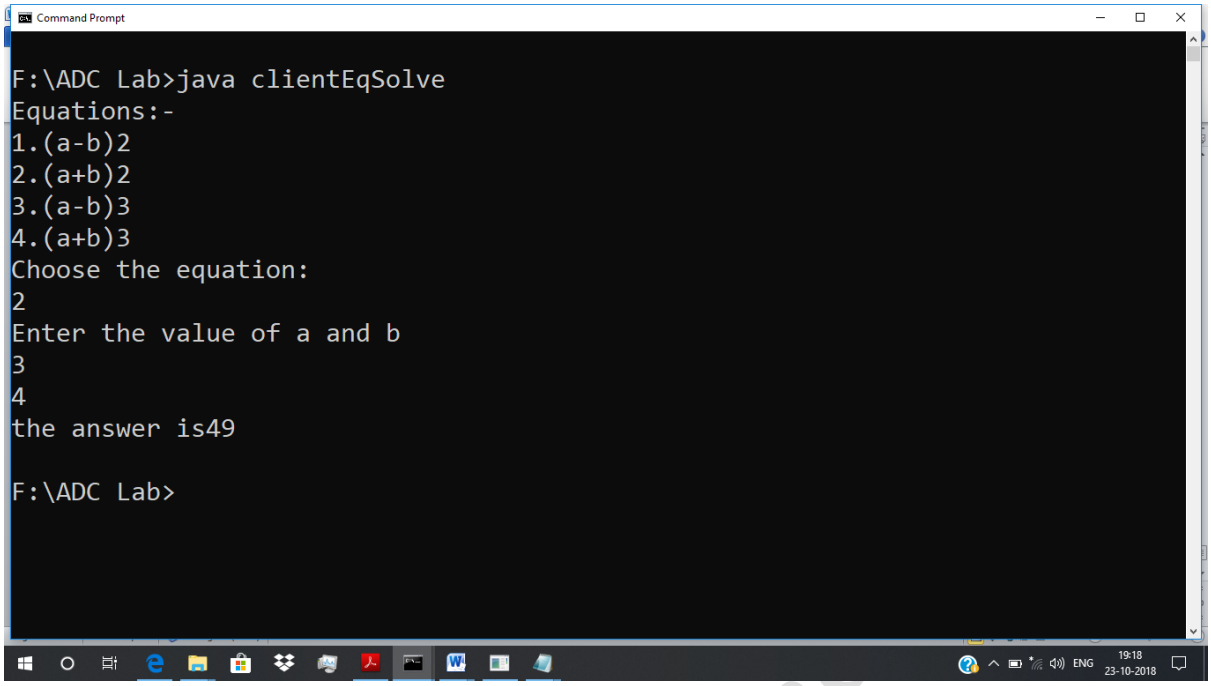
(iv) clientEqSolve.java

```
import java.io.*;
import java.net.*;
import java.rmi.*;
public class clientEqSolve
{
 public static void main(String[]args)
{
try
{
int num1,num2,res=0,choice;
intfEqSolve object=(intfEqSolve)Naming.lookup("hello");
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Equations:-");
System.out.println("1.(a-b)2");
System.out.println("2.(a+b)2");
System.out.println("3.(a-b)3");
System.out.println("4.(a+b)3");
System.out.println("Choose the equation:");
choice=Integer.parseInt(br.readLine());
System.out.println("Enter the value of a and b");
num1=Integer.parseInt(br.readLine());
num2=Integer.parseInt(br.readLine());
switch(choice) {

case 1:
res=object.solveEq1(num1,num2);
break;
case 2:
```

```
res=object.solveEq2(num1,num2);
break;
case 3:
res=object.solveEq3(num1,num2);
break;
case 4:
res=object.solveEq4(num1,num2);
break;
default:
System.out.println("Invalid option");
break;
}
System.out.println("the answer is"+res);
}
catch(Exception e)
{
System.out.println("Exception:"+e);
} } }
```

Output :





Prog. 16. Write a program to increment counter in shared memory.

Code :

SharedMemoryServer.java

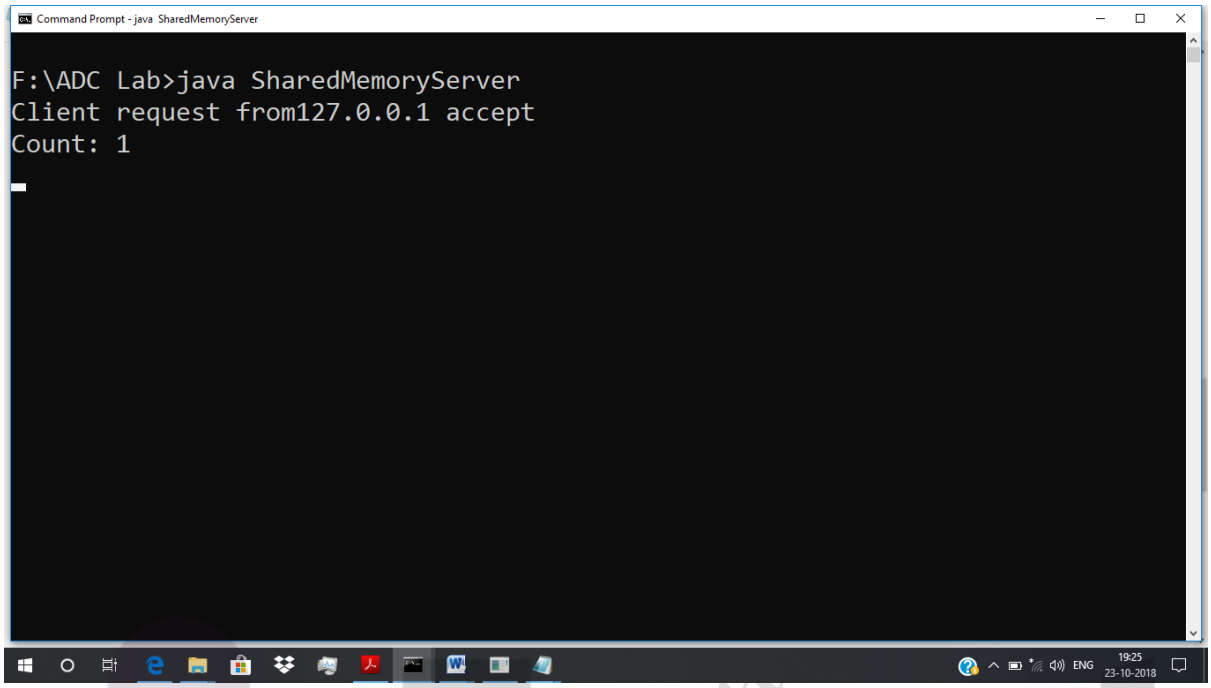
```
/* Implementation of Shared Memory */
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.logging.*;
public class SharedMemoryServer
{

   static int a = 50;
   static int count = 0;
  public static int getA(PrintStream cout)
 {
   count ++;
     --a;
   cout.println(a);
   return a;
 }
public void setA(int a)
{
  this.a = a;
 }
 public static void main(String args[]) throws Exception
{
   int x,y;
   String op;
   ServerSocket ss = new ServerSocket(2000);
   while(true)
   {
   Socket sk = ss.accept();
   BufferedReader cin = new BufferedReader(new InputStreamReader(sk.getInputStream()));
PrintStream cout = new PrintStream(sk.getOutputStream());
System.out.println("Client request from"+sk.getInetAddress().getHostAddress()+" accept");
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
String s;
s = cin.readLine();
Scanner sc = new Scanner(s);
op = sc.next();
if(op.equalsIgnoreCase("show"))
{
  x = getA(cout);
}
else
{
```

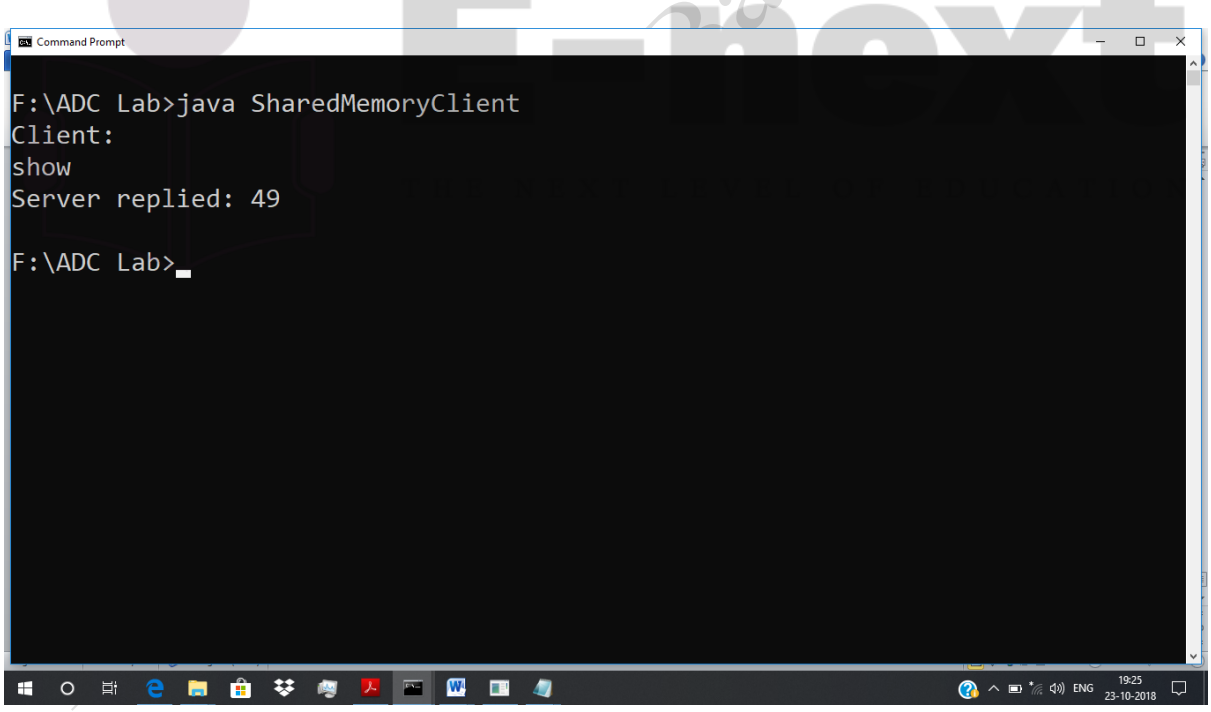
```
  cout.println("Check Syntax");
  break;
}
System.out.println("Count: "+count);
sk.close();
cin.close();
cout.close();
stdin.close();
} // close while loop
ss.close();
}
}
```

SharedMenoryClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class SharedMemoryClient
{
  private static Object host;
public static void main(String args[] ) throws Exception
{
   BufferedReader sin;
   PrintStream sout;
   BufferedReader stdin;
   Socket sk = new Socket("127.0.1.1",2000);
   sin = new BufferedReader(new InputStreamReader(sk.getInputStream()));
sout = new PrintStream(sk.getOutputStream());
stdin = new BufferedReader(new InputStreamReader(System.in));
String s;
while(true)
{
   System.out.println("Client: ");
   s = stdin.readLine();
   sout.println(s);
   s = sin.readLine();
   System.out.println("Server replied: "+s);
    break;
 }
  sin.close();
  sout.close();
  stdin.close();
}
}
```

Output:





Note : Open Multiple Client window to get new counter value per client window.

Prog. 17. Remote Objects for Database Access.

Retrieve Database Table records from Client by passing remote object to Server using RMI.

Code :

(i) IntDB.java

```
import java.rmi.*;
public interface IntDB extends Remote
{
  public String getData(String s, String db) throws RemoteException;
}
```

(ii) DBImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;

public class DBImpl extends UnicastRemoteObject implements IDb
{
   String sql,str=" ";
    public DBImpl() throws RemoteException
   {   }
 public String getData(String sql )
 {
     this.sql = sql;
     try
     {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection("jdbc:odbc:STD1");
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(sql);
    while(rs.next())
    {
    int r = rs.getInt(1);
    String n = rs.getString(2);
     int p = rs.getInt(3);
     int c = rs.getInt(4);
     int m = rs.getInt(5);
       str = str+" "+r+" "+n+" "+p+" "+c+" "+m+"\n";
  }

   }catch(Exception e) { }
  return str;
 }
}
```

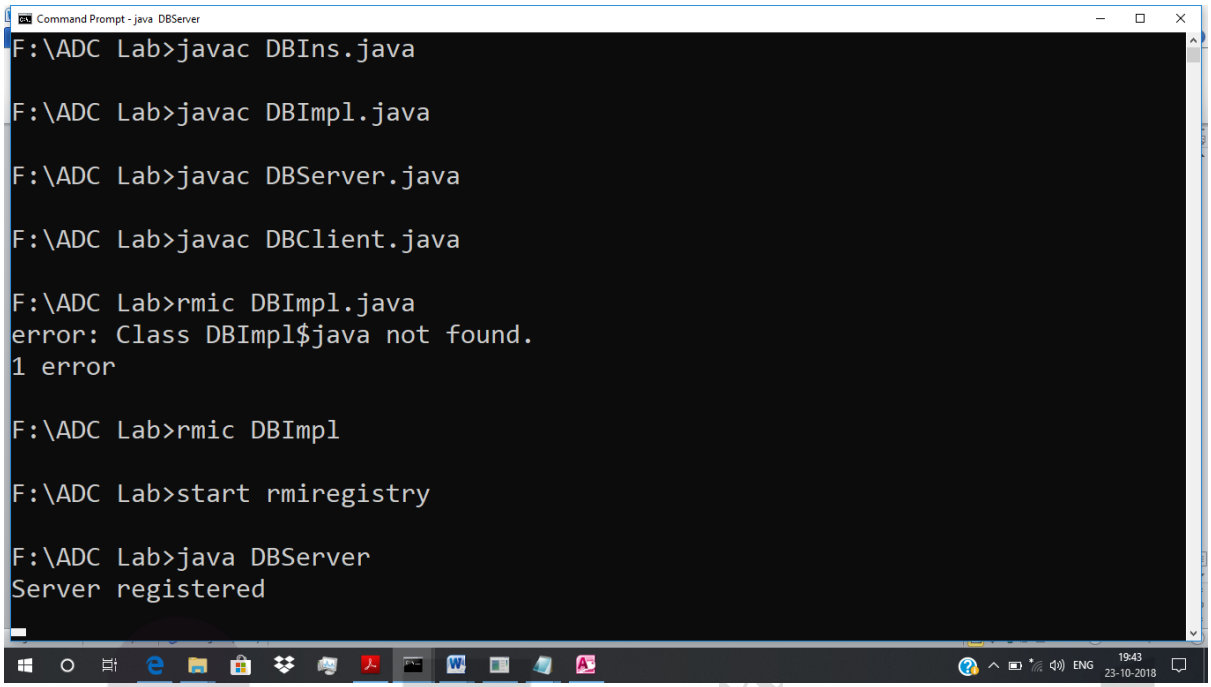
(iii) DBServer.java

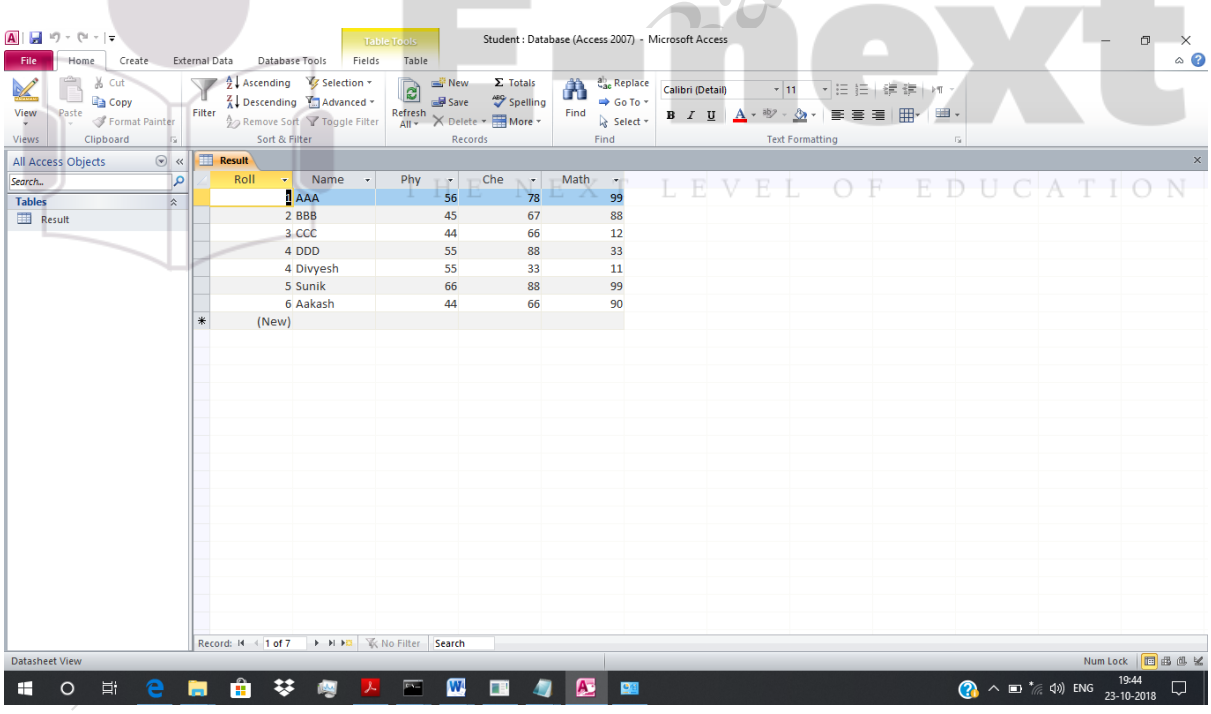
```
import java.rmi.*;
import java.net.*;
public class DBServer
{
public static void main(String args[]) throws Exception
{
  DBImpl dbi = new DBImpl();
  Naming.rebind("DBI",dbi);
 System.out.println("Server registered");
 }
}
```

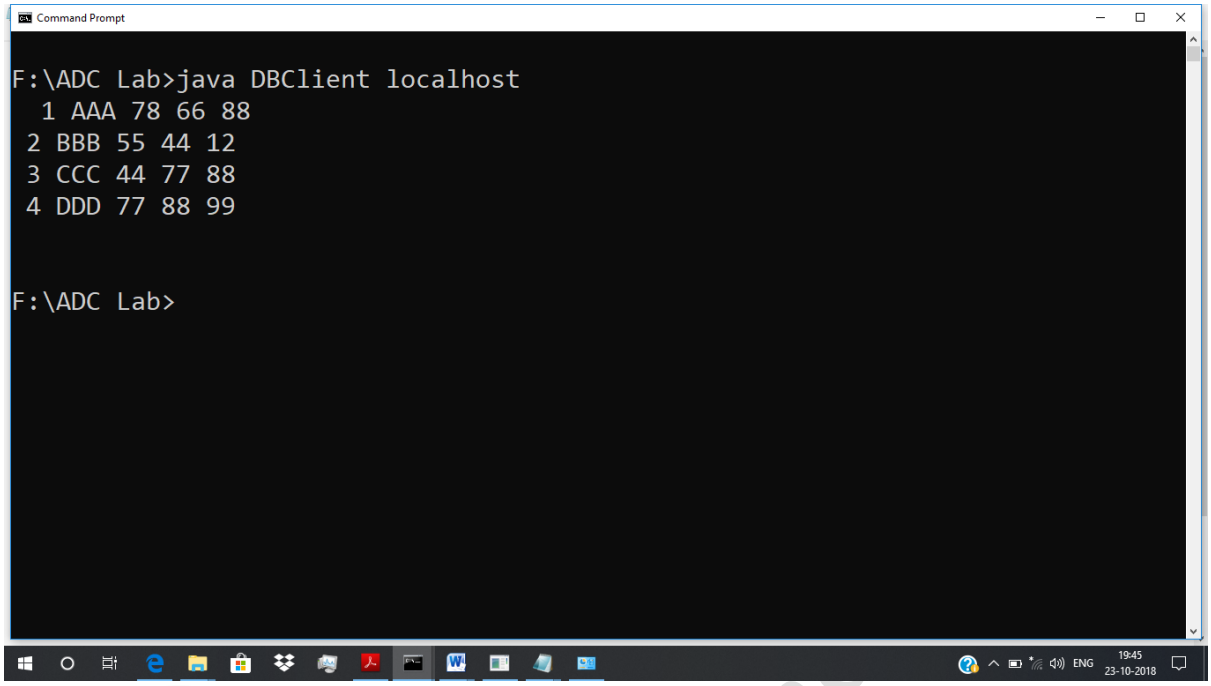
(iv) DBClient.java

```
import java.rmi.*;
import java.net.*;
public class DBClient
{
 public static void main(String args[]) throws Exception
{
   String url = "rmi://"+args[0]+"/DBI";
  IDb id = (IDb)Naming.lookup(url);
  String res = id.getData("select * from Result");
  /* */

  System.out.println(res);

 }
}
```

Output:


Command Prompt - java DBServer
F:\ADC Lab>javac DBIns.java
F:\ADC Lab>javac DBImpl.java
F:\ADC Lab>javac DBServer.java
F:\ADC Lab>javac DBClient.java
F:\ADC Lab>rmic DBImpl.java
error: Class DBImpl$java not found.
1 error
F:\ADC Lab>rmic DBImpl
F:\ADC Lab>start rmiregistry
F:\ADC Lab>java DBServer
Server registered


Student : Database (Access 2007) - Microsoft Access
Result
Roll Name Phy Che Math
1 AAA 56 78 99
2 BBB 45 67 88
3 CCC 44 66 12
4 DDD 55 88 33
4 Divyesh 55 33 11
5 Sunik 66 88 99
6 Aakash 44 66 90
(New)



Prog. 18 Retrieve multiple records from multiple database using RMI

Code :

(i) IntDB.java

```
import java.rmi.*;
public interface IntDB extends Remote
{
  public String getData(String s, String db) throws RemoteException;
}
```

(ii) DBImplement.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.net.*;
import java.sql.*;
public class DBImplement extends UnicastRemoteObject implements IntDB
{
  String str,str1;
   public DBImplement() throws RemoteException
   {   }
  public String getData(String sql,String db)
  {
   String URL = "jdbc:odbc:STDMTL";
   try
   {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection con = DriverManager.getConnection(URL);
   Statement stmt = con.createStatement();
   ResultSet rs  = stmt.executeQuery(sql);
   ResultSetMetaData rsmd = rs.getMetaData();
    str = "  ";str1= " ";
   for(int i=1;i<=rsmd.getColumnCount();i++)
    {
      str1 = str1+rsmd.getColumnName(i)+"\t";
    }
   System.out.println("-----------------------------------");
  while(rs.next())
  {
    for(int i=1;i<=rsmd.getColumnCount();i++)
    {
      str = str+ rs.getString(i)+"\t";
     }
      str = str+"\n";
    }
 }catch(Exception e) { }
   return(str1+"\n"+str);
```

```
}
}
```

(iii) DBMultiServer.java

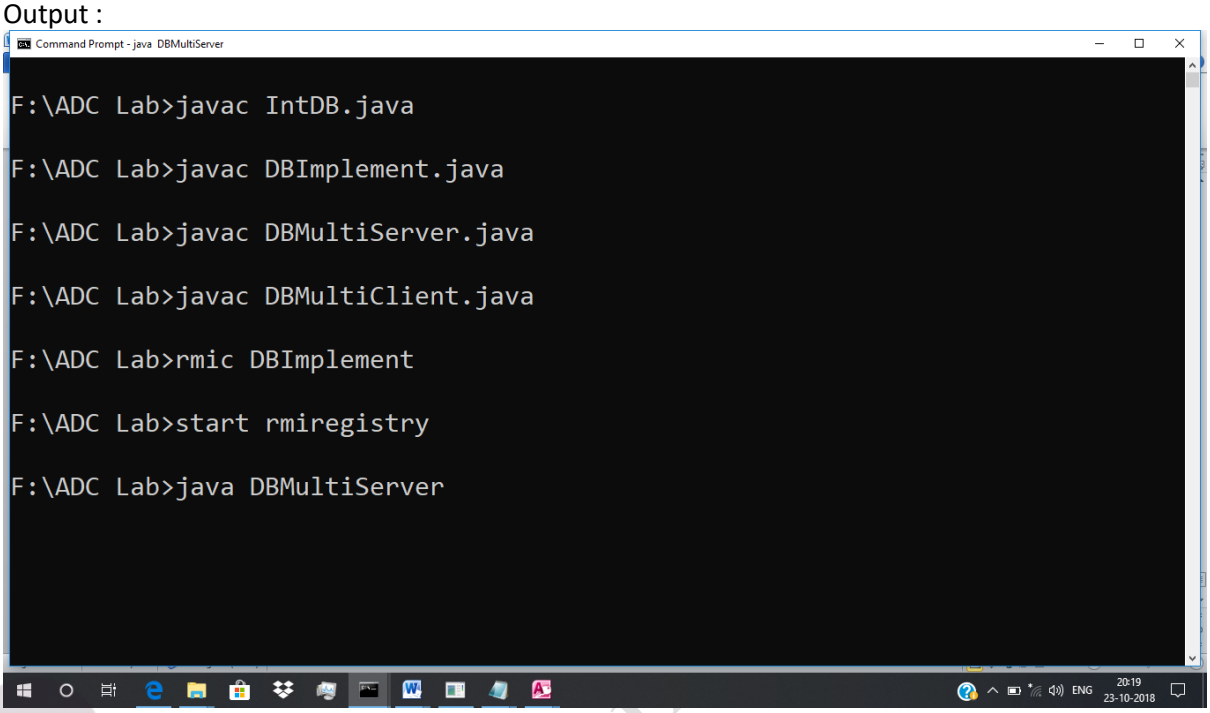
```
import java.rmi.*;
import java.net.*;

public class DBMultiServer
{
 public static void main(String args[] ) throws Exception
 {
    DBImplement dbi = new DBImplement();
    Naming.rebind("DBMServer",dbi);
 }
}
```

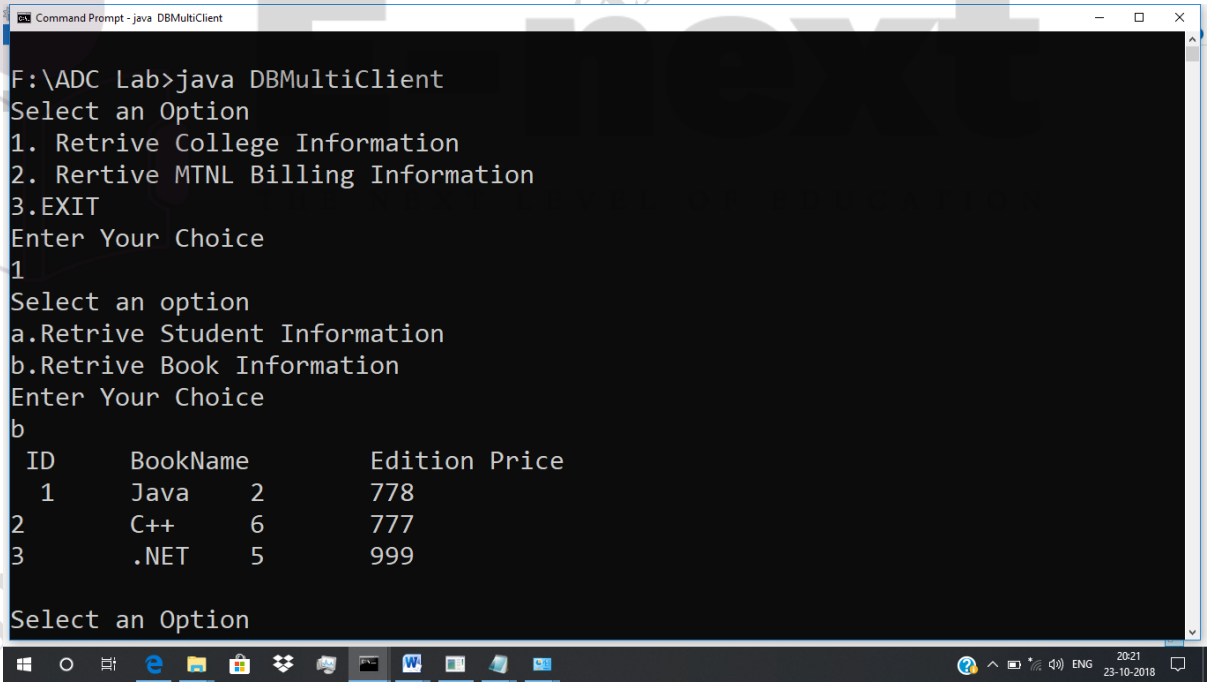
(iv) DBMultiClient.java

```
import java.rmi.*;
import java.net.*;
import java.io.*;
public class DBMultiClient
{
  public static void main(String args[])
 {
    String db=" ",sql=" ",ch=" ",ch1=" ",res=" ";
    try
    {
   BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 while(true)
 {
  System.out.println("Select an Option");
 System.out.println("1. Retrive College Information");
 System.out.println("2. Rertive MTNL Billing Information");
 System.out.println("3.EXIT");
 System.out.println("Enter Your Choice");
 ch = br.readLine();

 if(ch.equals("1"))
  {
     db = "ClgDB";
 System.out.println("Select an option");
 System.out.println("a.Retrive Student Information");
 System.out.println("b.Retrive Book Information");
 System.out.println("Enter Your Choice");
  ch1 = br.readLine();
  if(ch1.equals("a"))
```

```
  {
    sql = "select * from Student";
   }
   else if(ch1.equals("b"))
  {
    sql = "select * from Book";
   }
  else
  {
    System.out.println("Select valid option");
 }
} // Option 1 If close
 else if(ch.equals("2"))
 {
     db = "MTNLDB";
  }
 else if(ch.equals("3"))
 {
   System.exit(0);
 }
else
 {
   System.out.println("Select Valid Option");
 }
String url = "rmi://127.0.0.1/DBMServer";
IntDB id = (IntDB)Naming.lookup(url);
res = id.getData(sql,db);
System.out.println(res);
}
}catch(Exception e){ }
}
}
```

Output :


Command Prompt - java DBMultiServer
F:\ADC Lab>javac IntDB.java
F:\ADC Lab>javac DBImplement.java
F:\ADC Lab>javac DBMultiServer.java
F:\ADC Lab>javac DBMultiClient.java
F:\ADC Lab>rmic DBImplement
F:\ADC Lab>start rmiregistry
F:\ADC Lab>java DBMultiServer


Command Prompt - java DBMultiClient
F:\ADC Lab>java DBMultiClient
Select an Option
1. Retrive College Information
2. Rertive MTNL Billing Information
3.EXIT
Enter Your Choice
1
Select an option
a.Retrive Student Information
b.Retrive Book Information
Enter Your Choice
b
ID BookName Edition Price
1 Java 2 778
2 C++ 6 777
3 .NET 5 999
Select an Option

Prog. 19 : Simple program for addition of two number using Stateless Session Bean.

Code :

(i) CalBean.java

```
import javax.ejb.Local;

/**
 *
 * @author Vikram
 */
@Local
public interface CalBeanLocal {

   Integer sum(int a, int b);

}
```

(ii) Cal.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <form action="CalServlet" method="get">
      First Number <input type="text" name="n1"><br>
      Second Number <input type="text" name="n2"><br>
      <input type="submit" value="ADD">
    </form>
  </body>
</html>
```

(iii) Stateless Session Bean Program to implements Business Method add.java

```
package demo;

import javax.ejb.Stateless;

/**
 *
 * @author Vikram
 */
@Stateless
public class CalBean implements CalBeanLocal {

   @Override
   public Integer sum(int a, int b) {
      return (a+b);
```

```
    }

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")

}
```

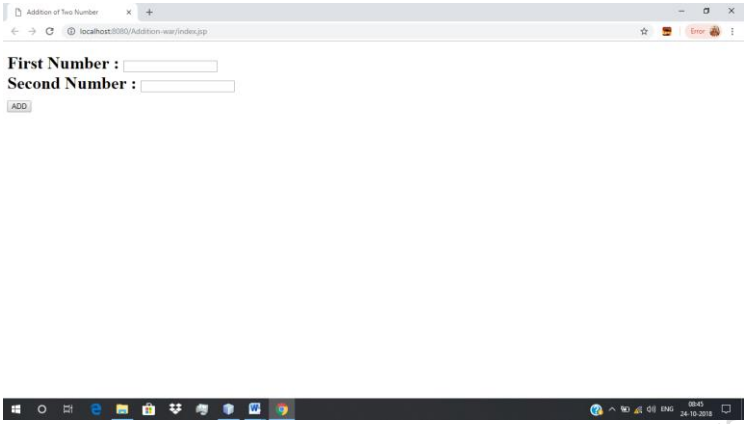
(iv) Servlet program to call Business Method

```
package aaa;

import demo.CalBeanLocal;
import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author Vikram
 */
public class CalServlet extends HttpServlet {

    @EJB
    private CalBeanLocal calBean;


    /**
     * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
     * methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet CalServlet</title>");
```

```
            out.println("</head>");
            out.println("<body>");
            int a = Integer.parseInt(request.getParameter("n1"));
            int b = Integer.parseInt(request.getParameter("n2"));
            out.println("<h1>Addition is " +calBean.sum(a, b) + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the +
sign on the left to edit the code.">
    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
```

}

Output :



Prog. 20 : Sample Program for Basic arithmetic operations implemented in Session Bean.

(i) Cal.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1> Simple Calculator</h1>
    <form method="get" action="AddServlet">
      First Number <input type="text" name="n1"><br>
      Second Number <input type="text" name="n2"><br>
      <input type="Submit" name="add" value="ADD">
      <input type="Submit" name="sub" value="SUB">
      <input type="Submit" name="mul" value="MUL">
    </form>
  </body>
</html>
```

(ii) CalBeanLocal.java

```
package demo.bean;

import javax.ejb.Local;

/**
 *
 * @author Vikram
 */
@Local
public interface CalBeanLocal {

    Integer sum(int x, int y);

    Integer sub(int m, int n);

    Integer mul(int x, int y);

}
```

(iii) Stateless Session Bean for Calculator CalBean.java

```
package demo.bean;

import javax.ejb.Stateless;

/**
 *
 * @author Vikram
 */
@Stateless
public class CalBean implements CalBeanLocal {

    @Override
    public Integer sum(int x, int y) {
        return (x+y);
    }

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")

    @Override
    public Integer sub(int m, int n) {
        return (m-n);
    }

    @Override
    public Integer mul(int x, int y) {
        return (x*y);
    }


}
```

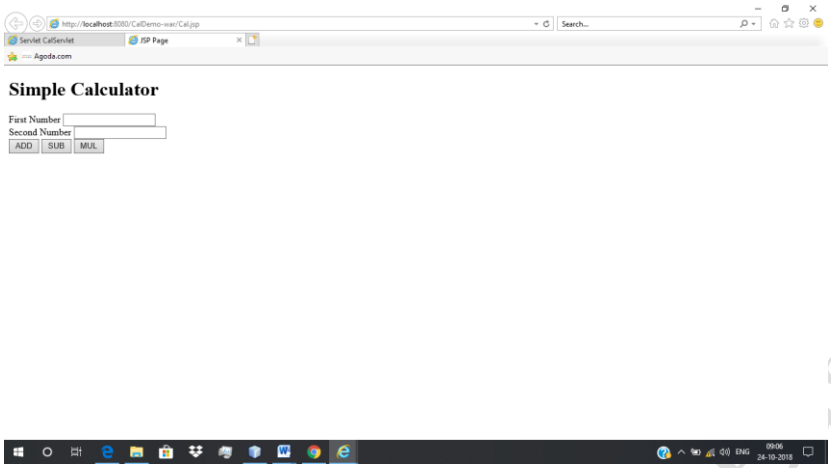
(iv) Servlet Program to call Business Method

```
package dd;

import demo.bean.CalBeanLocal;
import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
```

```
 * @author Vikram
 */
public class AddServlet extends HttpServlet {

    @EJB
    private CalBeanLocal calBean;


    /**
     * Processes requests for both HTTP <code>GET</code> and
<code>POST</code>
     * methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet AddServlet</title>");
            out.println("</head>");
            out.println("<body>");
            int a = Integer.parseInt(request.getParameter("n1"));
            int b = Integer.parseInt(request.getParameter("n2"));
            String act1 = request.getParameter("add");
            String act2 = request.getParameter("sub");
            String act3 = request.getParameter("mul");
            if(act1!=null)
            {
            out.println("<h1>Addition is " + calBean.sum(a, b) + "</h1>");
            }
            if(act2!=null)
            {
               out.println("<h1> Substraction is"+calBean.sub(a, b)+"</h1>");
            }
            if(act3!=null)
            {
               out.println("<h1> Multiplication  is"+calBean.mul(a, b)+"</h1>");
            }
            out.println("</body>");
            out.println("</html>");
```

```
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on
the + sign on the left to edit the code.">
    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>

}
```

Output :



Prog. 21 : Sample Program for Message Driven Bean

Code:

(i)

index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<form action="mservlet" method="post">
        Your Message:<input type="text" name="txt1" />
<input type="submit" value="submit" />
</form>
</body>
</html>
```

(ii)

mservlet.java

```
package bean;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.annotation.Resource;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class mservlet extends HttpServlet {
   @Resource(mappedName = "jms/dest")
   private Queue dest;
   @Resource(mappedName = "jms/queue")
```

```
    private ConnectionFactory destFactory;
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException, JMSException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String str=request.getParameter("txt1");
        sendJMSMessageToDest(str);
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet mservlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>"+str+"</h1>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        try {
            processRequest(request, response);
        } catch (JMSException ex) {
            Logger.getLogger(mservlet.class.getName()).log(Level.SEVERE, null, ex);
        } }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        try {
            processRequest(request, response);
        } catch (JMSException ex) {
            Logger.getLogger(mservlet.class.getName()).log(Level.SEVERE, null, ex);
        } }
    @Override
    public String getServletInfo() {
    return "Short description";}
    private Message createJMSMessageForjmsDest(Session session, Object messageData) throws
JMSException {
        TextMessage tm = session.createTextMessage();
        tm.setText(messageData.toString());
        return tm;
```

```
    }
    private void sendJMSMessageToDest(Object messageData) throws JMSException {
        Connection connection = null;
        Session session = null;
        try {
            connection = destFactory.createConnection();
            session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageProducer messageProducer = session.createProducer(dest);
            messageProducer.send(createJMSMessageForjmsDest(session, messageData));
        } finally {
            if (session != null) {
                try {
                    session.close();
                } catch (JMSException e) {
                    Logger.getLogger(this.getClass().getName()).log(Level.WARNING, "Cannot close session",
e);
                }
            }
            if (connection != null) {
                connection.close();
            } }}}
```

(iii) Bean Program

NewMessagebean.java

```
package bean;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
@MessageDriven(mappedName = "jms/dest", activationConfig = {
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
acknowledge"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue")
})
public class NewMessageBean implements MessageListener {
    public NewMessageBean() {}
    @Override
    public void onMessage(Message message) {
        TextMessage t=null;
```

```
        t=(TextMessage)message;
        try {
           System.out.println(t.getText());} catch (JMSException ex) {
Logger.getLogger(NewMessageBean.class.getName()).log(Level.SEVERE, null, ex);  }  }}
```

Output:

Prog. 22 . Sample program to display the Employee name and salary using Entity bean

CODE:

(i) Index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<form action="newser" method="post">
<table border="1">
<thead>
</thead>
<tbody>
<tr>
<td>Name:=</td>
<td><input type="text" name="txt1" value="" /></td></tr><tr>
<td>Sal:=</td>
<td><input type="text" name="txt2" value="" /></td></tr><tr>
<input type="submit" value="submit">
</tr></tbody></table></form></body>
</html>
```

(ii) Employee.java

```
package bean;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class employee implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
    private String name;
    private int salary;
   public Long getId() {
      return id;
```

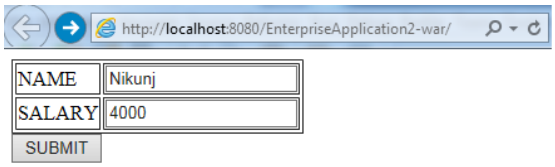
```
}
public void setId(Long id) {
   this.id = id;
}
@Override
public int hashCode() {
   int hash = 0;
   hash += (id != null ? id.hashCode() : 0);
   return hash;
}
@Override
public boolean equals(Object object) {
        if (!(object instanceof employee)) {
      return false;
   }
   employee other = (employee) object;
   if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
      return false;
   }
 return true;}
@Override
public String toString() {
   return "bean.employee[ id=" + id + " ]";
}
public String getName() {
   return name;
}
public void setName(String name) {
   this.name = name;
}

public int getSalary() {
   return salary;
}
public void setSalary(int salary) {
   this.salary = salary;
}}
```

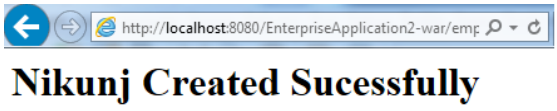
(iii) Newser.java

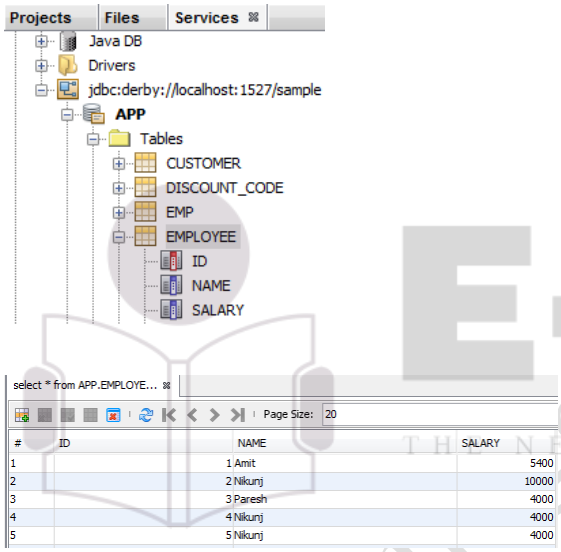
```
package bean;
import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class newser extends HttpServlet {
    @EJB
    private employeeFacadeLocal employeeFacade;
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        employee e=new employee();
        e.setName(request.getParameter("txt1"));
        e.setSalary(Integer.parseInt(request.getParameter("txt2")));
        employeeFacade.create(e);
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet newser</title>");
            out.println("</head>");
            out.println("<body>");
out.println("<h1>"+request.getParameter("txt1")+"Successfully </h1>");
            out.println("<h1>"+request.getParameter("txt2")+" </h1>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        } }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Output:







Prog. 23 : Implementation of mutual exclusion using Token Ring communication Process.

Code :

1)
TokenServer.java

```
import java.net.*;
import java.io.*;
public class TokenServer
{
	public static DatagramSocket ds;
	public static DatagramPacket dp;
	public static void main(String[] args)throws Exception
	{
		try
		{
			ds=new DatagramSocket(1000);
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
		while(true)
		{
			byte buff[]=new byte[1024];
	ds.receive(dp=new DatagramPacket(buff,buff.length));
	String str=new String(dp.getData(),0,dp.getLength());
	System.out.println("Message from " +str);
		}
	}
}
```

2) TokenClient2.java

```
import java.net.*;
import java.io.*;
public class TokenClient1
{
	public static DatagramSocket ds;
	public static DatagramPacket dp;
	public static BufferedReader br;
	static int cp=100;
	public static void main(String[] args)throws Exception
	{
```

```
boolean hasToken;
try
{
ds=new DatagramSocket(100);
}
catch(Exception e)
{
e.printStackTrace();
}
hasToken=true;
while(true)
{
        if(hasToken==true)
        {
System.out.println("Do you want to enter data...(yes/no):");
br=new BufferedReader(new InputStreamReader(System.in));
String ans=br.readLine();
if(ans.equalsIgnoreCase("yes"))
{
                                System.out.println("ready to send");
                                System.out.println("sending");
                                System.out.println("Enter the data");
br=new BufferedReader(new InputStreamReader(System.in));
String str="Client-1===> "+br.readLine();
byte buff[]=new byte[1024];
                                buff=str.getBytes();
ds.send(new DatagramPacket(buff,buff.length,InetAddress.getLocalHost(),1000));
                                System.out.println("now sending");
}
        else if(ans.equalsIgnoreCase("no"))
        {
                                System.out.println("I am busy state");
        //sending msg to client-2
                                        String msg="Token";
                                        byte bf1[]=new byte[1024];
                                        bf1=msg.getBytes();
                                        ds.send(new
DatagramPacket(bf1,bf1.length,InetAddress.getLocalHost(),200));
                                        hasToken=false;
                                //recevingmsg from client-2
                                        byte bf2[]=new byte[1024];
                                        ds.receive(dp=new
DatagramPacket(bf2,bf2.length));
```

```
                                        String clientmsg=new
String(dp.getData(),0,dp.getLength());
                                        System.out.println("The data is "+clientmsg);
                                        if(clientmsg.equals("Token"))
                                                hasToken=true;
                                System.out.println("I am leaving busy state");
        }
        }
        else
        {
System.out.println("Entering in receive mode.");
byte bf[]=new byte[1024];
ds.receive(dp=new DatagramPacket(bf,bf.length));
String clientmsg1=new String(dp.getData(),0,dp.getLength());
System.out.println("The data is "+clientmsg1);
if(clientmsg1.equals("Token"));
{
                                hasToken=true;
        }
}
}
}
}
```

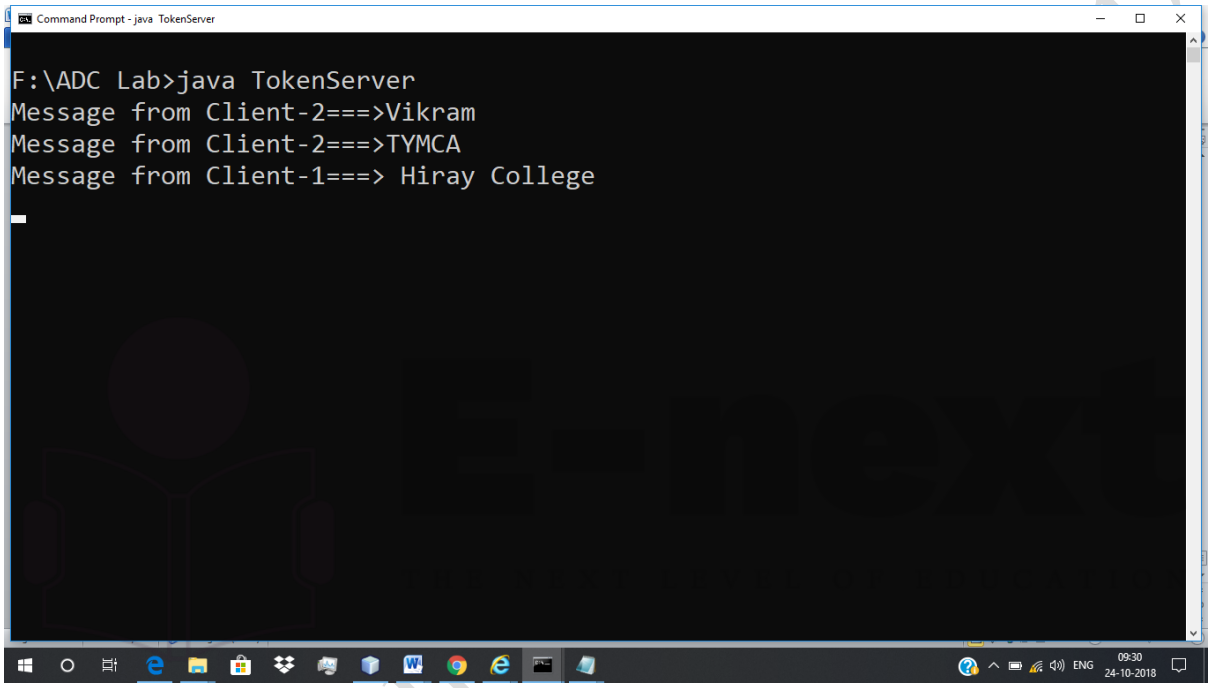
3) TokenClient3.java

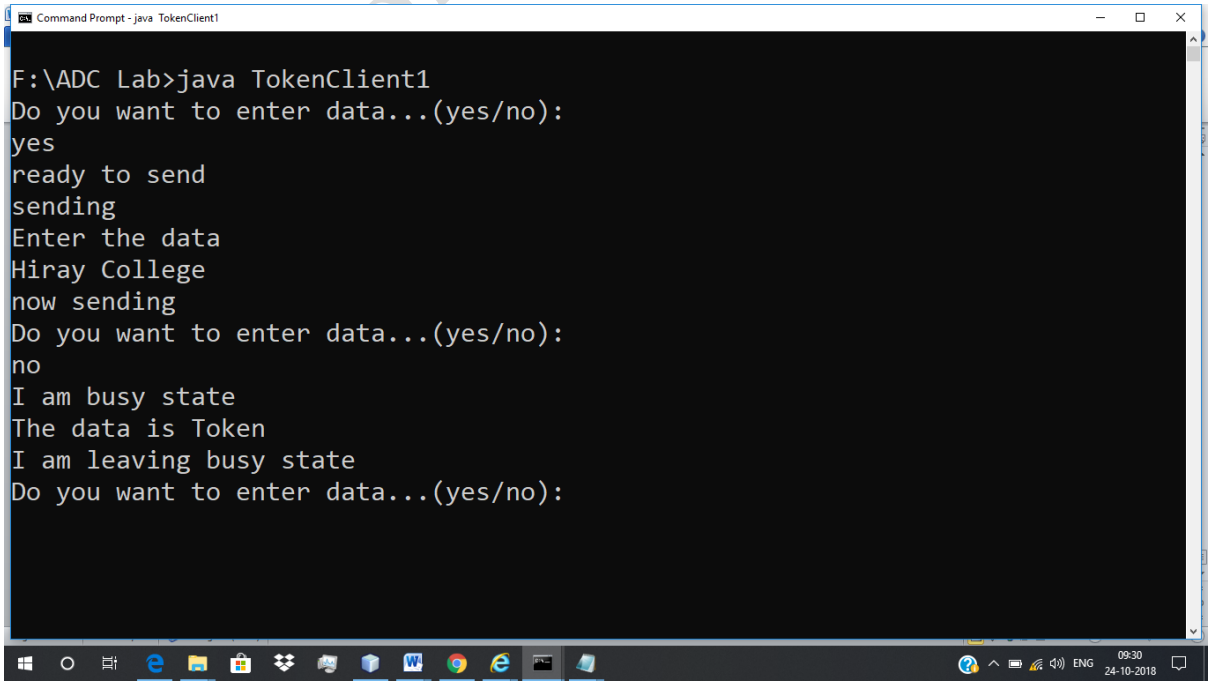
```
import java.net.*;
import java.io.*;
public class TokenClient2
{
        static DatagramSocket ds;
        static DatagramPacket dp;
        static BufferedReader br;
        public static void main(String[] args)throws Exception
        {
                try
                {
                        ds=new DatagramSocket(200);
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
                boolean hasToken=true;
```

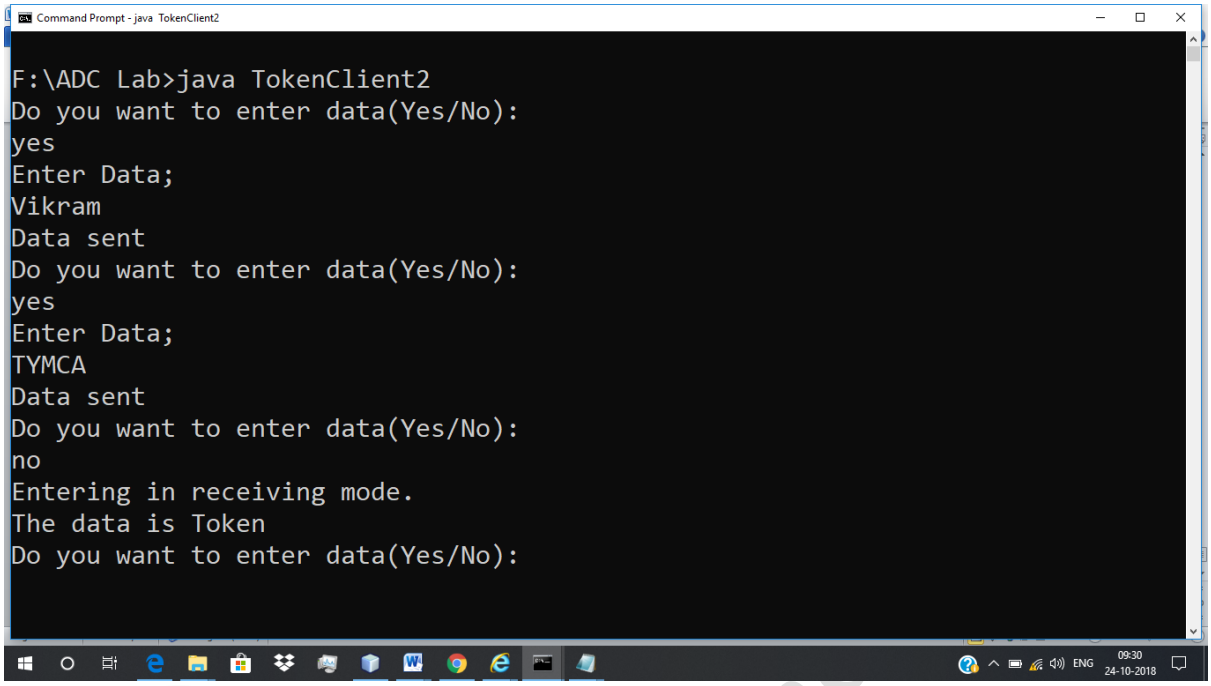
```
                while(true)
                {
                        //System.out.println("Entering if");
                        if(hasToken==true)
                        {
        System.out.println("Do you want to enter data(Yes/No):");
br=new BufferedReader(new InputStreamReader(System.in));
String str=br.readLine();
if(str.equalsIgnoreCase("yes"))
{
                                        System.out.println("Enter Data; ");
br=new BufferedReader(new InputStreamReader(System.in));
String msg="Client-2===>"+br.readLine();
byte bf1[]=new byte[1024];
                                        bf1=msg.getBytes();
ds.send(new DatagramPacket(bf1,bf1.length,InetAddress.getLocalHost(),1000));
                                        System.out.println("Data sent");
        }
        else
        {
        //send to client 1.
        String clientmsg="Token";
        byte bf2[] = new byte[1024];
                                        bf2=clientmsg.getBytes();
        ds.send(new DatagramPacket(bf2,bf2.length,InetAddress.getLocalHost(),100));
                                        hasToken=false;

}

}

else
{
try
{
                byte buff[]=new byte[1024];
                                        System.out.println("Entering in receiving mode.");
ds.receive(dp=new DatagramPacket(buff,buff.length));
String clientmsg1=new String(dp.getData(),0,dp.getLength());
                                        System.out.println("The data is "+clientmsg1);
if(clientmsg1.equals("Token"))
                                                hasToken=true;
}
```

```
catch(Exception e)
{
                                        e.printStackTrace();
}
}
}
}
}
```

Output :







Prog. 24 Study of Cloud Virtualization Technologies.

Virtualization refers to running multiple virtual computers, or virtual machines, inside a single physical computer. While the basic idea of virtualization is old (dating back to mainframe computers in 1960s), it has become mainstream only in the last 10-15 years. Today, most new servers are virtualized.

- Hypervisor is the operating system running on actual hardware. Virtual machines run as processes within this operating system. Sometimes the hypervisor is called `Dom0` or `Domain 0`.
- Virtual machine is a virtual computer running under a hypervisor.
- Container is a lightweight virtual machine that runs under the same operating system instance (kernel) as the hypervisor. Essentially, a container is nothing but a group of processes with their own name space for process identifiers etc.
- Virtual network is a logical network within a server, or possibly extending to multiple servers or even multiple data centres.
- Virtualization software is software that implements virtualization on a computer. It can be part of an operating system (or a special version of an operating system) or an application software package.

Virtualization is the basis of modern cloud computing.

WHY VIRTUALIZE?

Virtualization drivers in recent years have included:

- More powerful hardware, allowing each machine to run multiple applications simultaneously
- Pressure to lower IT costs and simplify IT administration
- Need to manage large-scale installations and clusters, such as server farms
- Improved security, reliability, scalability, and device independence
- Ability to mix multiple operating systems on same hardware.
- **Virtualization** has become a critically important focus of the IT world in recent years. Virtualization technologies are used by countless thousands of companies to consolidate their workloads and to make their IT environments scalable and more flexible. If you want to learn cloud computing, you'll simply have to absorb the basic virtualization technology concepts at some point.
- This course will give you all the **fundamental concepts to understand how Virtualization works:** why it's so important and how we moved from Virtualization to cloud computing. As a beginner course, you will find how Virtualization helps companies and professionals achieving better TCO and how it works from a technical point of view. Learn what is an hypervisor, how virtual machines are separated inside the same physical host and how they communicate with lower hardware levels. If you want to start a career in the cloud computing industry, you will need to know how the most common virtualization technologies works and how they are used in cloud infrastractures.
- A consistent part of this course is dedicated to the description of the most common technologies like: VMware, XEN, KVM and Microsoft Hyper-V. You will learn how they are used in the most common public cloud infrastructures and when to use them based on your needs.

Prog. 25 : Study of Cloud Data Center

Datacenter Design and Interconnection Networks
Datacenter Growth and Cost Breakdown : A large datacenter may be built with ten thousands or more servers. Smaller ones are built with hundreds or thousands of servers. The costs to build and maintain datacenter servers are increasing over the years. To keep a datacenter running well, typically, only 30% costs are due to purchase of IT equipments (such as servers and disks, etc), 33% costs are attributed to chiller, 18% on UPS (uninterruptible power supply), 9% on CRAC (computer room air conditioning), and the remaining 7% due to power distribution, lighting, and transformer costs. Thus the cost to run a datacenter is dominated by about 60% in management and maintenance costs. The server purchase cost did not increase much with time. The cost of electricity and cooling did increase from 5% to 14% in 15 years.
Low-Cost Design Philosophy: High end switches or routers cost a lot of money. Thus using high-end network devices does not fit the economics of cloud computing. However, the cost only plays one side of the story. The other side is to provide more bandwidth. Given a fix number of budget, much more low-end commodity switches can be purchased than the high end devices. The large number of low-end commodity switches can provide network redundancies as well as much larger bandwidth. This is also the same story while choosing the large number of commodity x86 servers instead of small number of mainframes. While using the large number of low cost commodity switches, software developer should design the software layer for handling the network traffic balancing, fault tolerant as well as expandability. This is also the same on the server side. The network topology design must face such situation. Currently, near all the cloud computing datacenters are using the Ethernet as the fundamental network technology. 7.2.1 Warehouse-Scale Datacenter Design Figure 7.8 shows a programmer's view of storage hierarchy of a typical WSC. A server consists of a number of processor sockets, each with a multicore CPU and its internal cache hierarchy, local shared and coherent DRAM, and a number of directly attached disk drives. The DRAM and disk resources within the rack are accessible through the first-level rack switches (assuming some sort of remote procedure call API to them), and all resources in all racks are accessible via the cluster-level switch.
Consider a datacenter built with 2,000 servers, each with 8 GB of DRAM and four 1-TB disk drives. Each group of 40 servers is connected through a 1-Gbps link to a rack-level switch that has an additional eight 1-Gbps ports used for connecting the rack to the cluster-level switch. It was estimated by Barroso and Holzle [9] that the bandwidth available from local disks is 200 MB/s, whereas the bandwidth from off-rack disks is just 25 MB/s via the shared rack uplinks. On the other hand, total disk storage in the cluster is almost ten million times larger than local DRAM. A large application that requires many more servers than can fit on a single rack must deal with these large discrepancies in latency, bandwidth, and capacity. In a large scale datacenter, each component is relatively cheap and easily obtained from the commercial market. The components used datacenters are very different from those in building supercomputer systems. With a scale of thousands of servers, concurrent failure, either hardware failure or software failure, of tens of nodes is common. There are many failures that can happen in hardware, for example CPU failure, disk IO failure, and network failure etc. It is even quite possible that the whole datacenter does not work while facing the situation of power crash. And also, some failures are brought by software. The service and data should not be lost in failure situation. Reliable can be achieved by redundant hardware. The software must keep multiple copies of data in different location and keep the data accessible while facing hardware or software errors.
Datacenter Interconnection Networks A critical core design of a datacenter is the interconnection network among all servers in the datacenter cluster. This network design must meet five special requirements: low latency, high bandwidth, low cost,

support MPI communications, and faul-tolearnt. The design of an inter-server network must satisfy both point-to-point and collective communication patterns among all server nodes. Specific design considerations are given below: Application Traffic Support: The network topology should support all MPI communication patterns.Both point-to-point and collective MPI communications must be supported. The network should have high bi-section bandwidth to meet this requirement. For example, one-to-many communications are used for supporting distruibute file accesses. One can use one or a few servers as metadata master servers which need to communicate with slave server nodes in the cluster. To support the MapReduce programming paradigm, the network must be designed to perform the map and reduce functions (to be treated in Chapter 7) in high speed. In other word, the underline network structure should support various network traffic patterns demanded by user applications.

Network Expandability : The interconnection network should be expandable. With thousands or even hundreds of thousands of server nodes, the cluster network interconnection should be allow to expand, once more servers are added into a datacenter. The network topology should be restructured while facing such an expected growth in the future. Also the network should be designed to support load balancing and data movement among the servers. None of the links should become a bottleneck to slow down the application performance. The topology of the interconnection should avoid such bottlenecks.

Fault Tolerance and Graceful Degradation: The interconnection network should provide some mechanism to tolerate link or switch failures. Multiple paths should be established between any two server nodes in a datacenter. Fault tolerant of servers is achieved by replicating data and computing among redundant servers. Similar redundancy technology should apply to the network structure. Both software and hardware network redundancy apply to cope with potential failures. One the software side, the software layer should be aware of network failure. Packets forwarding should avoid using the broken links. The network support software drivers should handle this transparently without affecting the cloud operations.

Switch-centric Datacenter Design : Currently, there are two approaches to building datacenter-scale networks : One is switch-centric and the other is server-centric. In a switch-centric network, the switches are used to connect the server nodes. The switch centric design does not affect the server side. No modifications to the servers are needed. The server-centric design does modify the operating system running on servers. Special drivers are designed for relaying the traffic. Switches still have to be organized for achieving the connections.

Container Datacenter Construction: The datacenter module is housed in a container. The modular container design include the network gear, compute, storage, and cooling. Just plug-in power, network, and chilled water, the datacenter should work. One needs to increase the cooling efficiency by varying the water and air flow with better air flow management. Another concern is to meet the seasonal load requirements. The construction of the container-based dataceneter may start with one system (server), them move to rack system design and finally the container system. The staged development may take different amounts of time and demand an increasing cost. For example, building one server system may take a few hours in racking and networking. Building a rack of 40 servers may take a half day's effort.

Prog. 26 Study of Grid Services

The World Wide Web began as a technology for scientific collaboration and was later adopted for e-business. We foresee—and indeed are experiencing—a similar trajectory for Grid technologies. The scientific resource sharing applications that motivated the early development of Grid technologies include the pooling of expertise through collaborative visualization of large scientific data sets, the pooling of computer power and storage through distributed computing for computation ally demanding data analyses, and increasing functionality and availability by coupling scientific instruments with remote computers and archives.1,7 We expect similar applications to become important in commercial settings—initially for scientific and technical computing applications, where we can already point to success stories—and then for commercial distributed computing applications. However, we expect that rather than enhancing raw capacity, the most important role for Grid concepts in commercial computing will be to offer solutions to new challenges that relate to the construction of reliable, scalable, and secure distributed systems. These challenges derive from the current rush, driven by technology trends and commercial pressures, to decompose and distribute through the network previously monolithic host-centric services.

OPEN GRID SERVICES ARCHITECTURE Enterprise computing systems must increasingly operate within virtual organizations (VO) with similarities to the scientific collaborations that originally motivated Grid computing. Depending on the context, the dynamic ensembles of resources, services, and people that comprise a scientific or business VO can be small or large, short- or long-lived, single- or multi-institutional, and homogeneous or heterogeneous. Individual ensembles can be structured hierarchically from smaller systems and may overlap in membership. Regardless of these differences, VO application developers face common requirements as they seek to deliver QoS—whether measured in terms of common security semantics, distributed workflow and resource management, coordinated fail-over, problem determination services, or other metrics— across a collection of resources with heterogeneous and often dynamic characteristics. Service orientation The Open Grid Services Architecture (OGSA) 3 supports the creation, maintenance, and application of the service ensembles that VOs maintain. OGSA adopts a common representation for computational and storage resources, networks, programs, databases, and the like. All are treated as services—network-enabled entities that provide some capability through the exchange of messages. Arguably, we could use the term object instead, as in systems like SOS8 and Legion,9 but we avoid it because of its overloaded meaning and because OGSA does not require object-oriented implementations. Adopting this uniform service-oriented model makes all components of the environment virtual—although the model must be grounded on implementations of physical resources. This service-oriented view partitions the interoperability problem into two subproblems: the definition of service interfaces and the identification of protocols that can invoke a particular interface. A service-oriented view addresses the need for standard interface definition mechanisms, local and remote transparency, adaptation to local OS services, and uniform service semantics. A service-oriented view also simplifies virtualization through encapsulation of diverse implementations behind a common interface. Virtualization Virtualization enables consistent resource

access across multiple heterogeneous platforms. Virtualization also enables mapping of multiple logical resource instances onto the same physical resource and facilitates management of resources within a VO based on composition from lower-level basic services to form more sophisticated services— without regard for how these services are implemented. Virtualizing Grid services also underpins the ability to map common service semantic behavior seamlessly onto native platform facilities. This virtualization is easier if we can express service functions in a standard form, so that any implementation of a service is invoked in the same man ner. We adopt the Web Services Description Language (WSDL) for this purpose

Service semantics: The Grid service Our ability to virtualize and compose services depends on more than standard interface definitions. We also require standard semantics for service interactions so that, for example, we have standard mechanisms for discovering service properties and different services follow the same conventions for error notification. To this end, OGSA defines a Grid service—a Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgradeability. Grid services also address authorization and concurrency control. This core set of consistent interfaces, from which we implement all Grid services, facilitates the construction of hierarchal, higher-order services that can be treated uniformly across layers of abstraction. As Figure 1 shows, a set of interfaces configured as a WSDL portType defines each Grid service. Every Grid service must support the GridService interface; in addition, OGSA defines a variety of other interfaces for notification and instance creation. Of course, users also can define arbitrary application-specific interfaces. The Grid service's serviceType, a WSDL extensibility element, defines the collection of portTypes that a Grid service supports, along with some additional information relating to versioning.

Grid Services Tools

ASKALON Architecture ASKALON has been designed as a set of distributed Grid services (see Figure 1). The services are based on the OGSI-technology and expose a platform independent standard API, expressed in the standard Web Services Description Language (WSDL) [11]. Platform dependent and proprietary services are pre-installed on specific appropriate sites from where they can be remotely accessed in a portable way, via the Simple Object Access Protocol (SOAP) [46] over HTTP. By isolating platform dependencies on critical resources, extra flexibility in installing and managing the tools is achieved. Each tool provides its own graphical User Portal to be accessed in a friendly and intuitive way. The User Portals are light-weight clients, easy to be installed and managed by the end-users. User Portals reside on the user's local machine (e.g. a notebook) and provide gateways to performance tools by dynamically creating and connecting to remote services. ASKALON services can be persistent (e.g. Registry and Factory) or transient, as specified by OGSI. All services can be accessed concurrently by multiple clients, which is an essential feature in a Grid environment and enables tool interoperability. The Grid Security Infrastructure (GSI) [26] based on single signon, credential delegation, and Web services security [5] through XML digital signature and XML encryption is employed for

authentication across ASKALON User Portals and Grid services. Remote service instances are created by a general-purpose Factory service (see Section 3.2 using the information from the Service Repository (see Section 3.1). At the same time, the portals discover and bind to existing service instances using the Registry service, described in Section 3.3. Additionally, the Data Repository (see Section 3.4) with a common standard schema definition, stores and shares common performance and output data of the applications under evaluation. It thus provides an additional mode of integration and interoperability among tools. To increase reliability of the system by avoiding single point of failures, multiple Registry, Service, and Data Repository instances are replicated on multiple sites and run independently. An OGSI-based asynchronous event framework enables Grid services to notify clients about interesting system and application events. ASKALON services support both push and pull event models, as specified by the Grid Monitoring Architecture (GMA) [50]. Push events are important for capturing dynamic information about running applications and the overall Grid system on-the-fly and avoids expensive continuous polling. Pull events are crucial for logging important information, for instance in cases when tools like ZENTURIO run in off-line mode, with disconnected off-line users. ASKALON classifies the Grid sites on which the services can run into two categories (see Figure 1): (1) Compute sites are Grid locations where end applications run and which host services intimately related to the application execution. Such services include the Experiment Executor of ZENTURIO, in charge of submitting and controlling jobs on the local sites and the Overhead Analyzer of SCALEA, which transforms raw performance data collected from the running applications into higher-level more meaningful performance overheads. (2) Service sites are arbitrary Grid locations on which ASKALON services are pre-installed or dynamically created by using the Factory service