<u>Chat Server</u>

<u>ServerApp.java</u>

```java
import java.io.InputStream;
import java.net.ServerSocket;
import java.net.Socket;


public class ServerApp implements Runnable{

    /**
     * @param args
     */
    public static Socket s=null;
    public static int i=1;
    public static String clientName = "";



    @Override
    public void run() {
        // TODO Auto-generated method stub

        try
        {
            InputStream is = s.getInputStream();
            byte[] b = new byte[1024];
            is.read(b);
            clientName="";
            clientName = new String(b).trim();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        new ChatGUI(s,clientName);
    }

    public static void main(String[] args) throws Exception{
        // TODO Auto-generated method stub
        ServerSocket ss = new ServerSocket(8089);
        ServerApp sa = new ServerApp();
        Thread t;
        try{
            while(true){
                System.out.println("Waiting for client "+i);
                s = ss.accept();
                i++;
                t = new Thread(sa);
                t.start();
            }
        }catch (Exception e) {
            // TODO: handle exception
        }
        finally{
            ss.close();
        }
    }


}
```

ClientApp.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;


public class ClientApp {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception{
        // TODO Auto-generated method stub

        System.out.print("Enter your name:");
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String name = br.readLine();
        Socket s = new Socket("localhost",8089);
        OutputStream os = s.getOutputStream();
        os.write(name.getBytes());
        new ChatGUI(s,"Admin");
    }

}
```

ChatGUI.java

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.io.IOException;
import java.io.InputStream;
```

```java
import java.io.OutputStream;

import java.net.Socket;
import java.net.SocketException;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class ChatGUI extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;
    Socket s;
    JButton button;
    JTextArea ta1, ta2;
    String msg = "", title;
    JScrollPane scrollPane1, scrollPane2;
    InputStream is;
    OutputStream os;

    ChatGUI(Socket x, String str) {
        s = x;
        title = str;
        button = new JButton("SEND");
        ta1 = new JTextArea(5, 20);
        ta2 = new JTextArea(5, 20);
        ta1.setEditable(false);
        scrollPane1 = new JScrollPane(ta1);
        scrollPane2 = new JScrollPane(ta2);
        setLayout(new FlowLayout());
        add(scrollPane1);
        add(scrollPane2);
        add(button);
        button.addActionListener(this);
        setSize(300, 300);
        setVisible(true);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setTitle("Messenger " + title);
        try {
            is = s.getInputStream();
            os = s.getOutputStream();
        } catch (IOException ioe) {
        }

        try {
            chat();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @SuppressWarnings("deprecation")
    public void chat() throws Exception {
        while (true) {
            try {
                byte data[] = new byte[50];
                is.read(data);
                msg = new String(data).trim();
                ta1.append(title+": " + msg + "\n");
            } catch (SocketException se) {
                JOptionPane.showMessageDialog(this, "Disconnected from "+title);
                this.dispose();
                Thread.currentThread().stop();
            }
        }
    }

    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
        msg = ta2.getText();
        try {
            os.write(msg.getBytes());
        } catch (IOException ioe) {
            // TODO Auto-generated catch block
            ioe.printStackTrace();
        }
        ta1.append("I: " + msg + "\n");
        ta2.setText("");
    }
}
```

RMI

RMIDemoInterface.java

```java
import java.rmi.*;

public interface RMIDemoInterface extends Remote{
    public int fact(int a) throws RemoteException;
}
```

RMIDemoClient.java

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

```
public class RMIDemoClient{
    public static void main(String[] args)throws Exception{
        String url="//localhost/rmiDemoObject";
        RMIDemoInterface obj= (RMIDemoInterface)Naming.lookup(url);

        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter any number:");
        int a=Integer.parseInt(br.readLine());

        System.out.println("The factorial is:"+obj.fact(a));

    }
}
```

RMIDemoServer.java

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

public class RMIDemoServer{
    public static void main(String[] args)throws Exception{
        RMIDemoInterface rmiDemoObject =new RMIDemoImpl();
        LocateRegistry.createRegistry(1099);
        Naming.rebind("rmiDemoObject",rmiDemoObject);
    }
}
class RMIDemoImpl extends UnicastRemoteObject implements RMIDemoInterface{

    RMIDemoImpl() throws RemoteException{
        super();
    }

    @Override

    public int fact(int a)throws RemoteException{
        int x=1;

        for(int i=1;i<=a;i++)x*=i;

        return x;

    }

}
```

DNS using Rmi

DnsClient.java

```
//DnsClient
import java.awt.FlowLayout;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.rmi.Naming;

public class DnsClient extends JFrame implements ActionListener {

    JButton b1, b2, b3, b4, b5;
    JPanel p1, p2;
    JLabel l1, l2;
    JTextField t1, t2;
    DataOutputStream output;
    DataInputStream input;

    DnsClient() {
        b1 = new JButton("AddHost");
        b2 = new JButton("Lookup");
        b3 = new JButton("Remove");
        b4 = new JButton("Refresh");
        b5 = new JButton("Close");
        p1 = new JPanel();
        p2 = new JPanel();
        l1 = new JLabel("Host");
        l2 = new JLabel("IP");
        t1 = new JTextField("", 20);
        t2 = new JTextField("", 20);
        p1.setLayout(new FlowLayout());
        p2.setLayout(new FlowLayout());

        p1.add(l1);
        p1.add(t1);
        p1.add(l2);
        p1.add(t2);

        p2.add(b1);
        p2.add(b2);
        p2.add(b3);
        p2.add(b4);
        p2.add(b5);
        add(p1, "North");
        add(p2, "South");
        setSize(600, 300);
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
```

```java
        b4.addActionListener(this);
        b5.addActionListener(this);
        setTitle("DNS Client Application");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        String s = e.getActionCommand();

        DnsRemoteInterface dri = null;
        try {
            dri = (DnsRemoteInterface) Naming
                    .lookup("rmi://localhost:1099/dnsrobj");

        } catch (Exception e1) {
            e1.printStackTrace();
        }
        if (s.equals("Refresh")) {
            t1.setText("");
            t2.setText("");
        }
        if (s.equals("Close")) {
            System.exit(0);
        }

        try {
            if (s.equals("AddHost")) {
                if (!t1.getText().trim().isEmpty()
                        || !t2.getText().trim().isEmpty()) {
                    Boolean b = dri.addHost(t1.getText(), t2.getText());
                    if (b == true) {
                        t2.setText("Registered");
                    } else {
                        t2.setText("Not Registered");
                    }
                } else {
                    JOptionPane.showMessageDialog(this,
                            "Fields cannot be blank");
                }
            }
            if (s.equals("Lookup")) {
                if (!t1.getText().trim().isEmpty()) {
                    String s1 = dri.lookupHost(t1.getText());
                    t2.setText(s1);
                    if (s1 == null) {
                        t2.setText("host name not found");
                    } else {
                        t2.setText("the ip address is " + s1);
                    }
                } else {
                    JOptionPane
                            .showMessageDialog(this, "Field cannot be blank");
                }
            }
            if (s.equals("Remove")) {
                if (!t1.getText().trim().isEmpty()) {
                    String s2 = dri.removeHost(t1.getText());
                    if (s2 == null) {
                        t2.setText("host name not found");
                    } else {
                        t2.setText("the ip address" + s2 + "is removed");
                    }
                } else {
                    JOptionPane
                            .showMessageDialog(this, "Field cannot be blank");
                }
            }
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        new DnsClient();
    }
}
```

DnsRemoteInterface.java

```
//DnsRemoteInterface
import java.rmi.Remote;

public interface DnsRemoteInterface extends Remote{
    public boolean addHost(String hostName,String hostIP) throws java.rmi.RemoteException;
    public String lookupHost(String hostName) throws java.rmi.RemoteException;
    public String removeHost(String hostName) throws java.rmi.RemoteException;

}
```

DnsServer.java

```
import java.io.*;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Properties;
```

```java
public class DnsServer extends UnicastRemoteObject implements DnsRemoteInterface{
    Properties hostRecords;
    FileInputStream fin = null;
    FileOutputStream fout = null;
    File nameList,dird;

    protected DnsServer() throws RemoteException {
        super();
        hostRecords = new Properties();
        dird = new File("d:/temp/");
        if (!dird.exists()) {
                dird.mkdir();
        }
        nameList = new File("d:/temp/NameList.txt");
        if (!nameList.exists()) {
            try {
                nameList.createNewFile();
            } catch (IOException e) {
            }
        }
        nameList.setReadOnly();

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

        DnsRemoteInterface robj = (DnsRemoteInterface)new DnsServer();
        System.out.println("Creating RMI Registry...");
        Registry reg = LocateRegistry.createRegistry(1099);
        System.out.println("Binding Remote Object...");
        reg.rebind("dnsrobj", robj);
        System.out.println("Remote Object bound.");
        System.out.println("\nPress Ctrl+C to stop.");
    }

    @Override
    public boolean addHost(String hostName, String hostIP)
            throws RemoteException {
        // TODO Auto-generated method stub
        hostRecords.clear();
        nameList.setWritable(true);
        try {
            fin = new FileInputStream(nameList);
            if (fin != null) {
                hostRecords.load(fin);
                fin.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (hostRecords.get(hostName) != null) {
            return false;
        }
        hostRecords.put(hostName, hostIP);
        try {
            fout = new FileOutputStream(nameList);
            hostRecords.store(fout, "");
            fout.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        nameList.setReadOnly();
        return true;

    }
    @Override
    public String lookupHost(String hostName) throws RemoteException {
        // TODO Auto-generated method stub
        String ip=null ;
        hostRecords.clear();
        try
            {
            fin = new FileInputStream(nameList);
            hostRecords.load(fin);
            ip = (String) hostRecords.get(hostName);
            fin.close();
            }
            catch (IOException ex) {
                ex.printStackTrace();
            }
            return ip;
    }
    @Override
    public String removeHost(String hostName) throws RemoteException {
        // TODO Auto-generated method stub
        String ip=null;
        hostRecords.clear();
        nameList.setWritable(true);
        try {
            fin = new FileInputStream(nameList);
            hostRecords.load(fin);
            ip = (String) hostRecords.remove(hostName);
            try {
                fout = new FileOutputStream(nameList);
                hostRecords.store(fout, "");
                fout.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
            nameList.setReadOnly();
            fin.close();

        } catch (Exception e) {
            e.printStackTrace();
            // TODO: handle exception
        }
        return ip;
    }
}
```

FTP

FTPClient.java

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.StringTokenizer;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;

public class FTPClient extends JFrame {

    private static final long serialVersionUID = 112345678L;

    JProgressBar jbar;
    JButton open, send, download, RefreshList;
    JFileChooser fc;
```

```java
    JLabel l, file;
    JPanel middle;
    String filenameonly;
    JList filelist;
    DefaultListModel model;
    JScrollPane scrollPane;

    public FTPClient(String name) {
        super(name);
        setLayout(new BorderLayout());
        setSize(600, 200);
        setResizable(false);
        // creating label
        l = new JLabel("Welcome");
        JPanel pj = new JPanel();
        pj.add(l);
        pj.setPreferredSize(new Dimension(600, 30));
        add(pj, BorderLayout.NORTH);

        // creating space for file
        middle = new JPanel();
        middle.setLayout(new BorderLayout());
        file = new JLabel("No File Selected");
        open = new JButton("open");
        open.addActionListener(new FOPENER());
        JPanel jp = new JPanel();
        jp.setLayout(new FlowLayout());
        jp.add(open);
        jp.setPreferredSize(new Dimension(100, 50));

        middle.add(jp, BorderLayout.EAST);
        JPanel jpfile = new JPanel();
        jpfile.setLayout(new FlowLayout());
        jpfile.add(file);
        jpfile.setPreferredSize(new Dimension(550, 50));
        middle.add(jpfile, BorderLayout.WEST);
        add(middle, BorderLayout.CENTER);

        JPanel bottom = new JPanel();
        bottom.setLayout(new BorderLayout());
        bottom.setPreferredSize(new Dimension(400, 200));

        JPanel jpsend = new JPanel();

        jpsend.setLayout(new FlowLayout());
        send = new JButton("upload");
        download = new JButton("Download");
        RefreshList = new JButton("Refresh List");
        jpsend.setPreferredSize(new Dimension(100, 200));
        jpsend.add(send);
        jpsend.add(download);
        jpsend.add(RefreshList);
        send.addActionListener(new SendFile());
        download.addActionListener(new DownloadFile());
        RefreshList.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent arg0) {
                // TODO Auto-generated method stub
                GetList();
            }
        });
        bottom.add(jpsend, BorderLayout.EAST);

        model = new DefaultListModel();
        filelist = new JList(model);
        filelist.setSelectionMode(ListSelectionModel.SINGLE_SELECTION );

        scrollPane = new JScrollPane(filelist);
        GetList();

        JPanel jppgbar = new JPanel();
        jppgbar.setLayout(new FlowLayout());
        jppgbar.add(scrollPane);
        bottom.add(jppgbar, BorderLayout.CENTER);
        add(bottom, BorderLayout.SOUTH);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }

    private void GetList() {
        // TODO Auto-generated method stub
        model.clear();
        try {
            dout.writeUTF("?");
            String s = din.readUTF();

            l.setText("Refershing List");
            StringTokenizer str = new StringTokenizer(s, "?");
            while (str.hasMoreTokens()) {
                model.addElement("       " + str.nextToken() + "       ");
            }
            l.setText("Refreshing List Completed");
        } catch (Exception e) {

        }
    }
    /**
     * @param args
     * @throws IOException
     * @throws UnknownHostException
     */
    static Socket ClientSoc;

    static DataInputStream din;
    static DataOutputStream dout;
    static BufferedReader br;

    public static void main(String[] args) throws UnknownHostException,
            IOException {
        // TODO Auto-generated method stub
        new FTPClient("Client");
        Socket soc = new Socket("127.0.0.1", 5217);
        ClientSoc = soc;
        din = new DataInputStream(ClientSoc.getInputStream());
        dout = new DataOutputStream(ClientSoc.getOutputStream());
        br = new BufferedReader(new InputStreamReader(System.in));
    }

    class FOPENER implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent arg0) {
            // TODO Auto-generated method stub
            fc = new JFileChooser();
            int rval = fc.showOpenDialog(FTPClient.this);
            if (rval == JFileChooser.APPROVE_OPTION) {
                file.setText(fc.getCurrentDirectory().toString() + "\\"
                        + fc.getSelectedFile().getName());
                filenameonly = fc.getSelectedFile().getName();
            } else {
                file.setText("No File Selected");
            }

        }

    }; // FOPENER

    class SendFile implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent arg0) {
            // TODO Auto-generated method stub
            String filename = file.getText();
            File f = new File(filename);

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

```java
            l.setText("File not Exists...");

            return;
        }

        try {
            dout.writeUTF(filenameonly);
            System.out.println(filename);

            din.readUTF();

            l.setText("Sending File ...");
            FileInputStream fin = new FileInputStream(f);
            int ch;
            do {
                ch = fin.read();
                dout.writeUTF(String.valueOf(ch));
            } while (ch != -1);
            fin.close();
            din.readUTF();
            l.setText("File send Sucessfully");
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
};

class DownloadFile implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent arg0) {
        // TODO Auto-generated method stub
        String i = (String) filelist.getSelectedValue();
        i = i.trim();

        if (i == null) {
            l.setText("Please Select a file");
            return;
        }

        try {
            dout.writeUTF("////" + i);
            String givenFilename = din.readUTF();
            System.out.println("given :"+givenFilename);
            if (!givenFilename.contentEquals(i)) {
                l.setText("The File " + i + "Doesn't Exist..");
                return;
            }
```

```
                File f = new File(i);
                l.setText("Downloading file..");

                dout.writeUTF("SendFile");
                FileOutputStream fout = new FileOutputStream(f);
                int ch;
                String temp;
                do {
                    temp = din.readUTF();
                    ch = Integer.parseInt(temp);
                    if (ch != -1) {
                        fout.write(ch);
                    }
                } while (ch != -1);
                fout.close();
                dout.writeUTF("OS");
                l.setText("File Downloaded");
            } catch (Exception e) {

            }
        }
    };

}; // class
```

<u>FTPServer.java</u>

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
```

```java
public class FTPServer extends JFrame {

    private static final long serialVersionUID = 112345678L;

    static JLabel l;

    JPanel middle;
    JList filelist;
    static DefaultListModel model;
    JScrollPane scrollPane;
    JButton refresh;

    public FTPServer(String name) throws IOException {
        super(name);
        setLayout(new BorderLayout());
        setSize(600, 200);
        setResizable(false);
        // creating label
        l = new JLabel("Waiting for Connection");
        JPanel pj = new JPanel();
        pj.add(l);
        pj.setPreferredSize(new Dimension(600, 30));
        add(pj, BorderLayout.NORTH);

        // creating space for file
        middle = new JPanel();
        // middle.setLayout(new BorderLayout());
        middle.setPreferredSize(new Dimension(600, 200));
        middle.setLayout(new BorderLayout());
        model = new DefaultListModel();

        filelist = new JList(model);
        filelist.setSelectionMode(ListSelectionModel.SINGLE_SELECTION );
        scrollPane = new JScrollPane(filelist);
        updateList();

        JPanel jscp = new JPanel();
        jscp.setLayout(new FlowLayout());
        jscp.add(scrollPane);

        middle.add(jscp, BorderLayout.CENTER);

        JPanel ref = new JPanel();
        ref.setLayout(new FlowLayout());
        refresh = new JButton("Refersh");
        refresh.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent arg0) {
                // TODO Auto-generated method stub
                try {
                    updateList();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });
        ref.add(refresh);
        middle.add(ref, BorderLayout.SOUTH);
        add(middle, BorderLayout.CENTER);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }

    private void updateList() throws IOException {
        // TODO Auto-generated method stub
        model.clear();
        File f = new File("."); // current directory

        File[] files = f.listFiles();
        for (File file : files) {
            if (file.isDirectory()) {
                continue;
            } else {
                model.addElement("       " + file.getName() + "       ");
            }

        }

    }

    /**
     * @param args
     * @throws IOException
     */

    static Socket ClientSoc;
    static DataInputStream din;
    static DataOutputStream dout;

    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
        ServerSocket soc = new ServerSocket(5217);
        FTPServer ftp = new FTPServer("Server");
        ClientSoc = soc.accept();
        l.setText("Connected");
        din = new DataInputStream(ClientSoc.getInputStream());
        dout = new DataOutputStream(ClientSoc.getOutputStream());
        Thread t = new Thread() {
            public void run() {
                try {
                    while (true) {
                        String filename = din.readUTF();
                        System.out.println("File name:"+filename  + filename.indexOf("_$_"));
                        if (filename.indexOf("?")==0) {

                            File f = new File("."); // current directory
                            String ans = "";
                            File[] files = f.listFiles();
                            for (File file : files) {
                                if (file.isDirectory()) {
                                    continue;
                                } else {
                                    ans += file.getName() + "?";
                                }

                            }
                            dout.writeUTF(ans);

                        } else if (filename.indexOf("////") == 0) {
                            String s = filename.substring(4);
                            System.out.println("REquested me to send"+s);

                            File f = new File(s);

                            if (!f.exists()) {
                                l.setText("Requested File not Found..." + s);
                                dout.writeUTF("???");
                                continue;
                            }

                            try {
                                dout.writeUTF(s);
                                System.out.println(s);

                                din.readUTF();

                                l.setText("Sending File ...");
                                FileInputStream fin = new FileInputStream(f);
                                int ch;
                                do {
                                    ch = fin.read();
                                    dout.writeUTF(String.valueOf(ch));
                                } while (ch != -1);
                                fin.close();
                                din.readUTF();
                                l.setText("File send Sucessfully");
                            } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                            }
                        } else {
                            System.out.println(filename);
                            l.setText("recivening file..");
                            File f = new File(filename);

                            dout.writeUTF("SendFile");
                            FileOutputStream fout = new FileOutputStream(f);
                            int ch;
                            String temp;
                            do {
                                temp = din.readUTF();
                                ch = Integer.parseInt(temp);
                                if (ch != -1) {
                                    fout.write(ch);
                                }
                            } while (ch != -1);
                            fout.close();
                            dout.writeUTF("OS");
                            l.setText("FileRecived");
                        }
                    }
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        };
        t.start();
    }
}
```

TCP

TCPClient.java

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

class TcpClient {
    public static void main(String[] args) throws Exception {
        System.out.println("connecting to server");
        Socket cs=new Socket("localhost",8088);

        BufferedReader br=new BufferedReader(new InputStreamReader( System.in));
        System.out.println("The Local Port "+cs.getLocalPort()+"\nThe Remote 
Port"+cs.getPort());
        System.out.println("The Local socket is "+cs);
        System.out.println("Enter your name");
        String str=br.readLine();
        //SENDING DATA TO SERVER
        OutputStream os=cs.getOutputStream();
        os.write(str.getBytes());
        //READING DATA FROM SERVER
        InputStream is=cs.getInputStream();
        byte data[]=new byte[50];
        is.read(data);
        //PRINTING MESSAGE ON CLIENT CONSLOE
        String mfs=new String(data);
        mfs=mfs.trim();
        System.out.println(mfs);
    }
}
```

TCPServer.java

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

public class TcpServer {
    public static void main(String[] args) throws Exception {
        ServerSocket ss=new ServerSocket(8088);
        System.out.println("server is ready!");
        Socket ls=ss.accept();
        while (true){
            System.out.println("Client Port is "+ls.getPort());
            //READING DATA FROM CLIENT
            InputStream is=ls.getInputStream();
            byte data[]=new byte[50];
            is.read(data);
            String mfc=new String(data);
            //mfc: message from client
            mfc=mfc.trim();
            String mfs="Hello:"+mfc;
            //mfs: message from server
            //SENDING MSG TO CLIENT
            OutputStream os=ls.getOutputStream();
            os.write(mfs.getBytes());
        }
    }
}
```

UDP

UDPClient.java

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

class UDPClient{
    public static void main(String[] args) throws Exception {
        byte[] buff=new byte[1024];
        DatagramSocket ds = new DatagramSocket(8089);
        DatagramPacket p=new DatagramPacket(buff,buff.length);

        BufferedReader br=new BufferedReader(new InputStreamReader(
            System.in));
        System.out.print("Enter your name:");
        String msg = br.readLine();
        buff = msg.getBytes();
        ds.send(new DatagramPacket(buff,buff.length, InetAddress.getLocalHost(),8088));
        ds.receive(p);
        msg = new String( p.getData(),0,p.getLength()).trim();
        System.out.println("Msg received "+msg);

    }
}
```

UDPServer.java

```java
import java.net.*;
class UDPServer{
    public static void  main(String[] args) throws Exception{
        byte buff[]=new byte[1024];
        DatagramSocket ds =new DatagramSocket(8088);
        DatagramPacket p=new DatagramPacket(buff,buff.length);

        System.out.println("Server ready :");

        ds.receive(p);
        String msg = new String( p.getData(),0,p.getLength()).trim();
        String str = "Hello "+new String(buff);
        buff = str.getBytes();
        ds.send(new DatagramPacket(buff,buff.length,InetAddress.getLocalHost(),8089));
        System.out.println("Msg received "+msg);
    }
}
```

RPC

Client

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

class cli {

   public static void main(String[] args) throws Exception {
      Socket sock = new Socket("127.0.0.1", 3000);
      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));
      System.out.println("Client ready, type and press Enter key");
      String receiveMessage, sendMessage, temp;
      while (true) {
         System.out.println("\nEnter operation to perform(add,sub,mul,div)....");
         temp = keyRead.readLine();
         sendMessage = temp.toLowerCase();
         pwrite.println(sendMessage);
         System.out.println("Enter first parameter :");
         sendMessage = keyRead.readLine();
         pwrite.println(sendMessage);
         System.out.println("Enter second parameter : ");
         sendMessage = keyRead.readLine();
         pwrite.println(sendMessage);
         System.out.flush();
         if ((receiveMessage = receiveRead.readLine()) != null) {
            System.out.println(receiveMessage);
         }
      }
   }
}
```

Server

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

   public static void main(String[] args) throws Exception {
      ServerSocket sersock = new ServerSocket(3000);
      System.out.println("Server ready");
      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, fun;
        int a, b, c;
        while (true) {
            fun = receiveRead.readLine();
            if (fun != null) {
                System.out.println("Operation : " + fun);
            }
            a = Integer.parseInt(receiveRead.readLine());
            System.out.println("Parameter 1 : " + a);
            b = Integer.parseInt(receiveRead.readLine());
            if (fun.compareTo("add") == 0) {
                c = a + b;
                System.out.println("Addition = " + c);
                pwrite.println("Addition = " + c);
            }
            if (fun.compareTo("sub") == 0) {
                c = a - b;
                System.out.println("Substraction = " + c);
                pwrite.println("Substraction = " + c);
            }
            if (fun.compareTo("mul") == 0) {
                c = a * b;
                System.out.println("Multiplication = " + c);
                pwrite.println("Multiplication = " + c);
            }
            if (fun.compareTo("div") == 0) {
                c = a / b;
                System.out.println("Division = " + c);
                pwrite.println("Division = " + c);
            }
            System.out.flush();
        }
    }

}
```

Write a program to increment Counter in Shared memory using JAVA.

```
import java.util.Scanner;

public class UnsynchronizedCounterTest {

static class Counter {

int count;

void inc() {

    count = count+1;

}

int getCount() {
    return count;
}
}
static Counter counter; // The counter that will be incremented.
static int numberOfIncrements; // Number of times each thread will increment it.
static class IncrementerThread extends Thread {
    public void run() {
        for (int i = 0; i < numberOfIncrements; i++) {
            counter.inc();
        }
    }
}
public static void main(String[] args) {
Scanner in = new Scanner(System.in); // For reading the user's inputs.
while (true) {
/* Get number of threads and number of increments per thread
* from the user. Exit if number of threads is <= 0. */
    System.out.println();
    System.out.print("How many threads do you want to run (Enter 0 to end)? ");
    int numberOfThreads = in.nextInt();
    if (numberOfThreads <= 0)
    break;

    do {
        System.out.println();
        System.out.println("How many times should each thread increment the counter? ");
        numberOfIncrements = in.nextInt();
        if (numberOfIncrements <= 0) {
            System.out.println("Number of increments must be positive.");
        }
    } while (numberOfIncrements <= 0);
System.out.println();
System.out.println("Using " + numberOfThreads + " threads.");
System.out.println("Each thread increments the counter "
+ numberOfIncrements + " times.");
/* Create the threads and start them. */
System.out.println();
System.out.println("Working...");
System.out.println();
IncrementerThread[] workers = new IncrementerThread[numberOfThreads];
counter = new Counter();
for (int i = 0; i < numberOfThreads; i++)
    workers[i] = new IncrementerThread();
for (int i = 0; i < numberOfThreads; i++)
    workers[i].start();
/* Wait for all threads to terminate. */
for (int i = 0; i < numberOfThreads; i++) {
    try {
        workers[i].join();
    }
    catch (InterruptedException e) {
    }
}
/* Display the results. */
System.out.println("The final value of the counter should be "
+ (numberOfIncrements*numberOfThreads));
System.out.println("Actual final value of counter is: " + counter.getCount());
System.out.println();
System.out.println();
} // end while
} // end main()
} // end class UnsynchronizedCounterTest
```

Write a program to Simulate the Distributed Mutual Exclusion.

```
#include<stdio.h>

#include<conio.h>

#include<dos.h>

#include<time.h>

void main()

{

int cs=0,pro=0;

double run=5;

char key='a';

time_t t1,t2;

clrscr();

printf("Press a key(except q) to enter a process into critical section.");

printf(" \nPress q at any time to exit.");

t1 = time(NULL) - 5;

while(key!='q')

{

while(!kbhit())

if(cs!=0)

{

t2 = time(NULL);

if(t2-t1 > run)

{

printf("Process%d ",pro-1);

printf(" exits critical section.\n");

cs=0;

}

}

key = getch();

if(key!='q')

{

if(cs!=0)
printf("Error: Another process is currently executing critical section Please wait till its
execution is over.\n");
else
{
printf("Process %d ",pro);

printf(" entered critical section\n");
cs=1;
pro++;
t1 = time(NULL);
}
}
}
}
```