1. Implement

a. Implement Echo client-server application in JAVA using TCP.

```
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());
        }
    }
}



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 CONSOLE
        String mfs=new String(data);
        mfs=mfs.trim();
        System.out.println(mfs);
    }
}
```

b. Implement a concurrent daytime client-server application in JAVA.

2. Implement

a. Implement Echo client-server application in JAVA using UDP.

```
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);

    }
}
```

```
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);
    }
}
```

b. Implement a concurrent daytime client-server application in JAVA.

3. Write a program to demonstrate Rikart-Agrawal Mutex (RAM) Mutual Exclusion in a distributed environment.

```
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.*;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
```

```
public class LamportMutex {
public class RicartAgrawalaMutex {
    public static volatile BlockingQueue<Message> messagesToBeProcessed
= new LinkedBlockingQueue<Message>();
    public static volatile List<Integer> replyPending;      //store
list of pending nodes from which reply hasnt been received
    public static volatile Comparator<RequestObject> comparatorForQueue
= new ComparatorForQueue ();
    public static volatile PriorityQueue<RequestObject> requestQueue =
new PriorityQueue<RequestObject>(50, comparatorForQueue);
    public static volatile Integer scalarClock = 0;
    public static volatile boolean isExecutingCS = false;
    public static boolean csEnter(){
        //init reply monitor
        replyPending = Collections.synchronizedList(new
ArrayList<Integer>(){
            public synchronized boolean add(int node){
                boolean ret = super.add(node);
                return ret;
            }
        });
        Iterator<Integer> itr =
CriticalSection.nodeMap.keySet().iterator();
        while(itr.hasNext()){
            replyPending.add(itr.next());
        }
        //replyPending.remove(CriticalSection.self.getNodeId());
        //end initialization of reply monitor
        //Critical section entry request sent
        CriticalSection.isRequestSent = true;
        //Scalar clock update
        scalarClock++;
        //Add my request to request queue
        RequestObject requestObject = new RequestObject(scalarClock,
CriticalSection.self.getNodeId());
        requestQueue.add(requestObject);
        //Send request message to all nodes
        Message request = new Message(MessageType.Request,
CriticalSection.self.getNodeId(),scalarClock);
        Iterator<Integer> iterator =
CriticalSection.nodeMap.keySet().iterator();
        try{
            while (iterator.hasNext()) {
                Node node =
CriticalSection.nodeMap.get(iterator.next());
                Socket socket = new Socket(node.getNodeAddr(),
node.getPort());
                ObjectOutputStream outMessage = new
ObjectOutputStream(socket.getOutputStream());
                outMessage.writeObject(request);
                socket.close();
            }
        }catch(Exception e){
            CriticalSection.isRequestSent = false;
            System.out.println("Exception in sending request message");
            e.printStackTrace();
            return false;
        }
        System.out.println("Sending Request time - " +
LamportMutex.scalarClock + " request Number - " +
CriticalSection.countRequestsSent);
        System.out.println("Sent Request time - " +
RicartAgrawalaMutex.scalarClock + " request Number - " +
CriticalSection.countRequestsSent);
        //Block enterCS function till isExecutingCS is not marked as
true
        while(true){
            if(isExecutingCS) {
                try {
                    CriticalSection.bufferedWriter.write("\n STARTING
CS BY - " + CriticalSection.self.getNodeId() + " AT TIME - " +
LamportMutex.scalarClock);
                    CriticalSection.bufferedWriter.write("\n STARTING
CS BY - " + CriticalSection.self.getNodeId() + " AT TIME - " +
RicartAgrawalaMutex.scalarClock);
                    CriticalSection.bufferedWriter.flush();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                break;
            }
        }
        return true;
    }
    public static void csExit(){
        //Mark isExecutingCS as false,
        isExecutingCS = false;
        //Remove yourself from requestQueue
        LamportMutex.requestQueue.poll();
        RicartAgrawalaMutex.requestQueue.poll();
        //Critical section entry request is not sent
        CriticalSection.isRequestSent = false;
        //Send release message to all nodes
        sendReleaseMessage();
        //Send reply message to all deferred requests
        sendReplyReleaseMessages();
    }

    public static void sendReleaseMessage() {
    public static void sendReplyReleaseMessages() {
        try{
            //Increment clock value
            LamportMutex.scalarClock = LamportMutex.scalarClock + 1;
            CriticalSection.bufferedWriter.write("\nRELEASE CS BY - " +
CriticalSection.self.getNodeId() + " AT TIME - " +
LamportMutex.scalarClock);
            RicartAgrawalaMutex.scalarClock =
RicartAgrawalaMutex.scalarClock + 1;
            CriticalSection.bufferedWriter.write("\nREPLY/RELEASE CS BY
- " + CriticalSection.self.getNodeId() + " AT TIME - " +
RicartAgrawalaMutex.scalarClock);
            CriticalSection.bufferedWriter.flush();
            //Generate release message
            Message releaseMessage = new Message(MessageType.Release,
CriticalSection.self.getNodeId(), LamportMutex.scalarClock);
            Iterator<Integer> iterator =
CriticalSection.nodeMap.keySet().iterator();
            Message releaseMessage = new Message(MessageType.Reply,
CriticalSection.self.getNodeId(), RicartAgrawalaMutex.scalarClock);
            Iterator<RequestObject> iterator =
RicartAgrawalaMutex.requestQueue.iterator();
            while (iterator.hasNext()) {
                Node node =
CriticalSection.nodeMap.get(iterator.next());
                Socket socket = new Socket(node.getNodeAddr(),
node.getPort());
                RequestObject requestObject = iterator.next();
                Socket socket = new
Socket(CriticalSection.nodeMap.get(requestObject.getNodeId()).getNodeAd
dr(),

CriticalSection.nodeMap.get(requestObject.getNodeId()).getPort());
                ObjectOutputStream outMessage = new
ObjectOutputStream(socket.getOutputStream());
                outMessage.writeObject(releaseMessage);
                socket.close();
            }
        }catch(Exception e){
            System.out.println("Exception in sending release message");
            System.out.println("Exception in sending reply/release
message");
            e.printStackTrace();
        }

        System.out.println("Release message sent to all neighbours");
        System.out.println("Reply/Release message sent to all
neighbours at time - " + RicartAgrawalaMutex.scalarClock);
    }

}
```

4. Develop a distributed chat server using TCP sockets in JAVA for a Single server- Single client environment.

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

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

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

import javax.swing.JButton;
import javax.swing.JFrame;
```

```java
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("");
    }
}
```

```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");
    }

}
```

```
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 = "";
    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();
        }
    }
    @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);
    }
}
```

5. Implement a distributed chat server using TCP sockets in JAVA for a Single server-Multiple client environment.

6. Write a program for Remote Method Invocation (RMI) mechanism for accessing remote methods (ADD, SUB, MUL & DIV).

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;


public class RMIDemoClient {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String url= "rmi://localhost:1099/rmiDemoObject";

        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        try {
            RMIDemoInterface remoteIntf = (RMIDemoInterface)
Naming.lookup(url);
            System.out.println(remoteIntf.sayHello());
            System.out.println("Enter two numbers:");
            System.out.print("a: ");
            int a = Integer.parseInt(br.readLine());
            System.out.print("b: ");
            int b = Integer.parseInt(br.readLine());
            int sum = remoteIntf.add(a, b);
            int deference = remoteIntf.subtract(a, b);
            int product = remoteIntf.multiply(a, b);

            System.out.println("The sum is : "+sum);
            System.out.println("The deference is : "+deference);
            System.out.println("The product is : "+product);


        } catch (MalformedURLException | RemoteException |
NotBoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (NumberFormatException e) {
            // TODO Auto-generated catch block
```

```java
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

```java
import java.rmi.Remote;
import java.rmi.RemoteException;


public interface RMIDemoInterface extends Remote{
    public String sayHello() throws RemoteException;
    public int add(int a, int b) throws RemoteException;
    public int subtract(int a, int b) throws RemoteException;
    public int multiply(int a, int b) throws RemoteException;
}
```

```java
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

class RMIDemoImpl extends UnicastRemoteObject implements
RMIDemoInterface{

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    protected RMIDemoImpl() throws RemoteException {
        super();
        // TODO Auto-generated constructor stub
    }

    @Override
    public String sayHello() throws RemoteException {
        // TODO Auto-generated method stub
        return "Hello Client! Welcome:";
    }

    @Override
    public int add(int a, int b) throws RemoteException {
        // TODO Auto-generated method stub
        return a+b;
    }

    @Override
    public int subtract(int a, int b) throws RemoteException {
        // TODO Auto-generated method stub
        return a-b;
    }

    @Override
    public int multiply(int a, int b) throws RemoteException {
        // TODO Auto-generated method stub
        return a*b;
    }

}
public class RMIDemoServer {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        try {
            RMIDemoInterface rmiDemoObject = new RMIDemoImpl();
            LocateRegistry.createRegistry(1099);
            Naming.rebind("rmiDemoObject",rmiDemoObject);
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

```
}
```

7. Write a program to calculate Factorial of the given number using the Remote Method Invocation (RMI) mechanism.

8. Write a program to perform Matrix NxN Multiplication using the Remote Method Invocation (RMI) mechanism.

9. Write a program for displaying Fibonacci Series using the Remote Method Invocation (RMI) mechanism.

10. Implement the Matrix Transportation program using the Remote Method Invocation (RMI) mechanism.

11. Write a program to perform Inverse of a Matrix using the Remote Method Invocation (RMI) mechanism.

12. Write program to search (any method) the number from a given list using the Remote Method Invocation (RMI) mechanism.

13. Write a program to sort the given list using the Remote Method Invocation (RMI) mechanism (any sorting technique)

14. Implement the String concatenation program using the Remote Method Invocation (RMI) mechanism.

15. Implement the program to reverse the given string using the Remote Method Invocation (RMI) mechanism.

16. Write a program illustrating Palindrome using the Remote Method Invocation (RMI) mechanism.

17. Develop a single client- Single server application that uses File Transfer Protocol (FTP) using 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.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;

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();
    }
}
```

```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;
    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();
```

```java
            } 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()) {
                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
```

18. Develop multiple clients- single server application that uses File Transfer Protocol (FTP) using JAVA.

19. Write a program for Remote Procedure Call (RPC) Protocol for accessing remote Procedure (ADD, SUB, MUL & DIV).

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();
        }
    }

}
```

20. Write a program for Remote Procedure Call (RPC) Protocol for accessing remote Procedure (NxN Matrix ADDITION).