Lab 1. Implement concurrent echo client server application in JAVA

TCP Server

```
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());
      }
   }
}
```

```
server is ready!
Client Port is 61268
Client Port is 61268
```

TCP Client

```
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());
      }
    }
}
```

```
connecting to server
The Local Port 61268
The RemotePort8088
The Local socket is Socket[addr=localhost/127.0.0.1,port=8088,localport=61268]
Enter your name
LocalHost 123
Hello:LocalHost 123
```

UDP Server

```
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());
      }
```

```
  }
}
```

```
Server ready :
Msg received LocalHost 123
```

UDP Client

```
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);
   }
}
```

```
Enter your name:LocalHost 123
Msg received Hello LocalHost 123
```

Lab 2. Implement a Distributed Chat Server using TCP Sockets in JAVA.

Server App

```
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);
    }
}
```

```
Waiting for client 1
Waiting for client 2
```

ClientApp

```
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");
    }
}
```

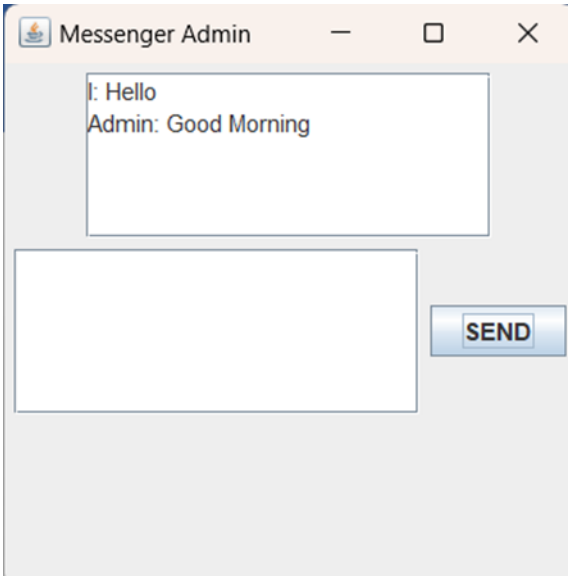
```
Enter your name:LocalHost 123
```

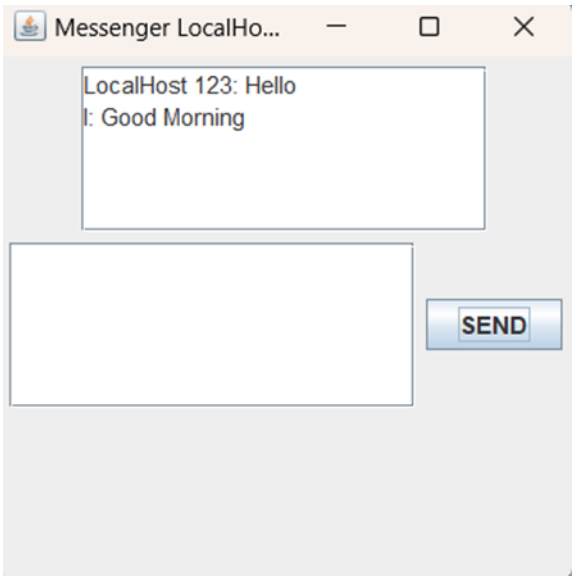
ChatGUI

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





Lab 3. Implement concurrent day-time client-server application in JAVA

Server_DT

```
import java.net.*;
import java.io.*;
import java.util.Date;
public class Server_DT {
   public static void main(String[] args) throws IOException {
      ServerSocket ss = new ServerSocket(5000);
      System.out.println("The server has reserved port No:" + ss.getLocalPort() + " for this Service");
      Socket cs = ss.accept();
      System.out.println(
            "Client with IP Address" + cs.getInetAddress() + "has communicated via port No:" +
cs.getPort());
      Date d = new Date();
      String s = "Current Date & Time on Server is:" + d;
      PrintWriter toclient = new PrintWriter(cs.getOutputStream(), true);
      toclient.print(s);
      toclient.close();
      cs.close();
      ss.close();
   }
}
```

```
The server has reserved port No:5000 for this Service
Client with IP Address/127.0.0.1has communicated via port No:61604
```

Client_DT

```
import java.net.*;
import java.io.*;
public class Client_DT {
   public static void main(String[] args) throws UnknownHostException, IOException {
      Socket cs = new Socket("LocalHost", 5000);
      System.out.println("Client" + cs.getInetAddress() + "is communicating from port No:" +
cs.getPort());
      BufferedReader fromserver = new BufferedReader(new
InputStreamReader(cs.getInputStream()));
      System.out.println(fromserver.readLine());
      fromserver.close();
      cs.close();
   }
}
```

```
ClientLocalHost/127.0.0.1is communicating from port No:5000
Current Date & Time on Server is:Thu Sep 28 20:15:17 IST 2023
```

Lab 4. Configure following options on server socket and tests them: SO KEEPALIVE, SO LINGER, SO_SNDBUF, SO RCV BUF, TCP_NODELAY

Server

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
public class Server {
   public static void main(String[] args) {
      try {
         // Create a server socket on port 12345
         ServerSocket serverSocket = new ServerSocket(12345);
         System.out.println("Server listening on port 12345...");
         // Accept client connection
         Socket clientSocket = serverSocket.accept();
         System.out.println("Client connected: " + clientSocket.getInetAddress().getHostAddress());
         // Enable SO_KEEPALIVE (TCP keep-alive)
         clientSocket.setKeepAlive(true);
         // Enable SO_LINGER with a linger time of 5 seconds
         clientSocket.setSoLinger(true, 5);
         // Set SO_SNDBUF to 64KB (64 * 1024 bytes)
         int sendBufferSize = 64 * 1024;
         clientSocket.setSendBufferSize(sendBufferSize);
         // Get input and output streams
         BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
         PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);
         // Read and send data
         String message;
         while ((message = reader.readLine()) != null) {
            System.out.println("Received from client: " + message);
            // Send response back to the client
            writer.println("Server response: " + message);
         }
         // Close the socket and server socket when done
         clientSocket.close();
         serverSocket.close();
      } catch (SocketException e) {
         // Handle socket-related exceptions
```

```
            e.printStackTrace();
        } catch (IOException e) {
            // Handle IO-related exceptions
            e.printStackTrace();
        }
    }
}
```

```
Server listening on port 12345...
Client connected: 127.0.0.1
Received from client: Hello, server!
```

Client

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.SocketException;
public class Client {
    public static void main(String[] args) {
        try {
            // Connect to the server on localhost:12345
            Socket socket = new Socket("localhost", 12345);
            // Enable SO_KEEPALIVE (TCP keep-alive)
            socket.setKeepAlive(true);
            // Enable SO_LINGER with a linger time of 5 seconds
            socket.setSoLinger(true, 5);
            // Set SO_SNDBUF to 64KB (64 * 1024 bytes)
            int sendBufferSize = 64 * 1024;
            socket.setSendBufferSize(sendBufferSize);
            // Get input and output streams
            BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
            // Send data to the server
            writer.println("Hello, server!");
            // Receive response from the server
            String response = reader.readLine();
            System.out.println("Received from server: " + response);
            // Close the socket when done
            socket.close();
        } catch (SocketException e) {
```

```
            // Handle socket-related exceptions
            e.printStackTrace();
        } catch (IOException e) {
            // Handle IO-related exceptions
            e.printStackTrace();
        }
    }
}
```

```
Received from server: Server response: Hello, server!
```

Lab 5. Write a program to Incrementing a counter in shared memory in JAVA

```
import java.util.Scanner;
public class UnsynchronizedCounterTest1 {
   static class Counter {
      int count;
      void inc() {
         count = count + 1;
      }
      int getCount() {
         return count;
      }
   }
   static Counter counter;
   static int numberofincrements;
   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);
      while (true) {
         System.out.println();
         System.out.println("how many threads do you want to run");
         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 (numberofthreads <= 0) {
               System.out.println("No of increments should be positive");
            }
         } while (numberofincrements <= 0);
         System.out.println();
         System.out.println("using" + numberofthreads + "threads");
         System.out.println("each thread increment the counter" + numberofincrements + "times");
         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();
        for (int i = 0; i < numberofthreads; i++) {
           try {
              workers[i].join();
           } catch (InterruptedException e) {
           }
        }
        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();
     }
  }
}
```

```
how many threads do you want to run
2

How many times should each thread increment the counter
5

using2threads
each thread increment the counter5times

working

the final value of the counter should be10
actual final value of counter is:10
```

Lab 7. Write a program to Implement Java RMI" mechanism for accessing methods of remote systems.

Add server

```
import java.rmi.*;
import java.net.*;
public class AddServer {
   public static void main(String args[]) {
      try {
         AddRemImpl locobj = new AddRemImpl();
         Naming.rebind("rmi:///AddRem", locobj);
      } catch (RemoteException re) {
         re.printStackTrace();
      } catch (MalformedURLException mfe) {
         mfe.printStackTrace();
      }
   }
}
```

Add client

```
import java.rmi.*;
import java.net.*;
import java.io.*;
import java.util.*;
public class AddClient {
   public static void main(String arges[]) {
      String host = "localhost";
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter first parameter");
      int a = sc.nextInt();
      System.out.println("Enter second parameter");
      int b = sc.nextInt();
      try {
         AddRemImpl remote = new AddRemImpl();
         AddRem remobj = (AddRem) Naming.lookup("rmi://" + host + "/AddRem");
         System.out.println(remote.addNum(a, b));
      } catch (RemoteException re) {
         re.printStackTrace();
      } catch (NotBoundException nbe) {
         nbe.printStackTrace();
      } catch (MalformedURLException mfe) {
         mfe.printStackTrace();
```

```
      }
   }
}
```

Add rem

```
import java.rmi.*;
public interface AddRem extends Remote {
   public int addNum(int a, int b) throws RemoteException;
}
```

Add rem implem

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class AddRemImpl extends UnicastRemoteObject implements
      AddRem {
   public AddRemImpl() throws RemoteException {
   }

   public int addNum(int a, int b) {
      return (a + b);
   }
}
```