I. Develop a client server application where a client types a message and presses the “Enter” key to send it to the server. The server responds to the message. Client should be prevented from sending the next message until it receives the response for the previous message. Your implementation should support not less than 3 clients active at the same time. This may be implemented either by using connectionless or connection oriented sockets.

Details of the clients connected such as IP address, port number have to be displayed at the server side.

Server:

**package** endsem;

**import** java.io.\*;

**import** java.text.\*;

**import** java.util.\*;

**import** java.net.\*;

**public** **class** Server

{

**public** **static** **void** main(String[] args) **throws** IOException

{

ServerSocket ss = **new** ServerSocket(5000);

**while** (**true**)

{

Socket s = **null**;

**try**

{

s = ss.accept();

System.***out***.println("Client connected: " + s);

DataInputStream dis = **new** DataInputStream(s.getInputStream());

DataOutputStream dos = **new** DataOutputStream(s.getOutputStream());

Thread t = **new** ClientHandler(s, dis, dos);

t.start();

}

**catch** (Exception e){

s.close();

e.printStackTrace();

}

}

}

}

**class** ClientHandler **extends** Thread

{

**final** DataInputStream dis;

**final** DataOutputStream dos;

**final** Socket s;

**public** ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos)

{

**this**.s = s;

**this**.dis = dis;

**this**.dos = dos;

}

@Override

**public** **void** run()

{

String received;

String toreturn;

**while** (**true**)

{

**try** {

received = dis.readUTF();

dos.writeUTF("Received message");

System.***out***.println(received);

} **catch** (IOException e) {

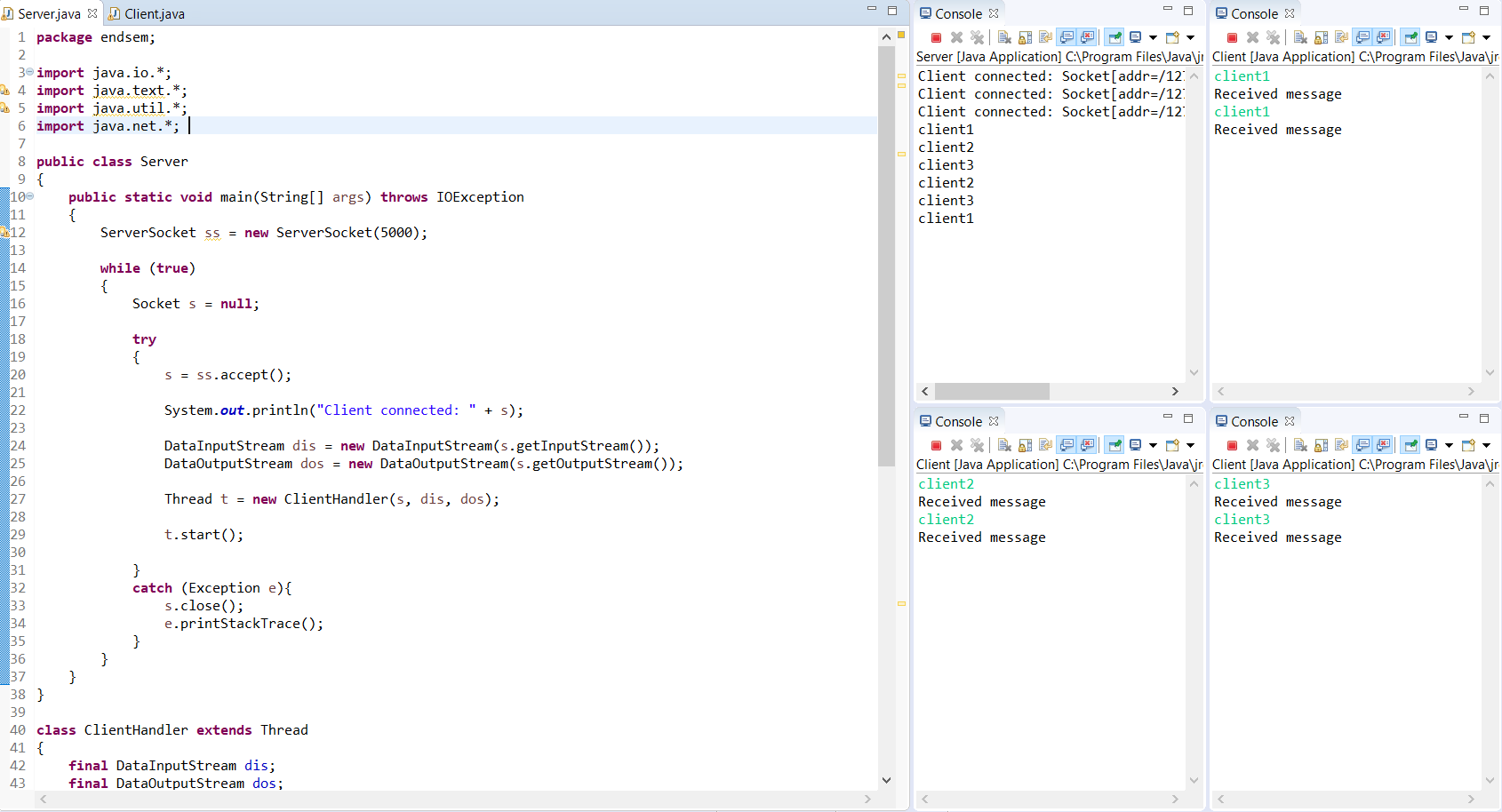
e.printStackTrace();

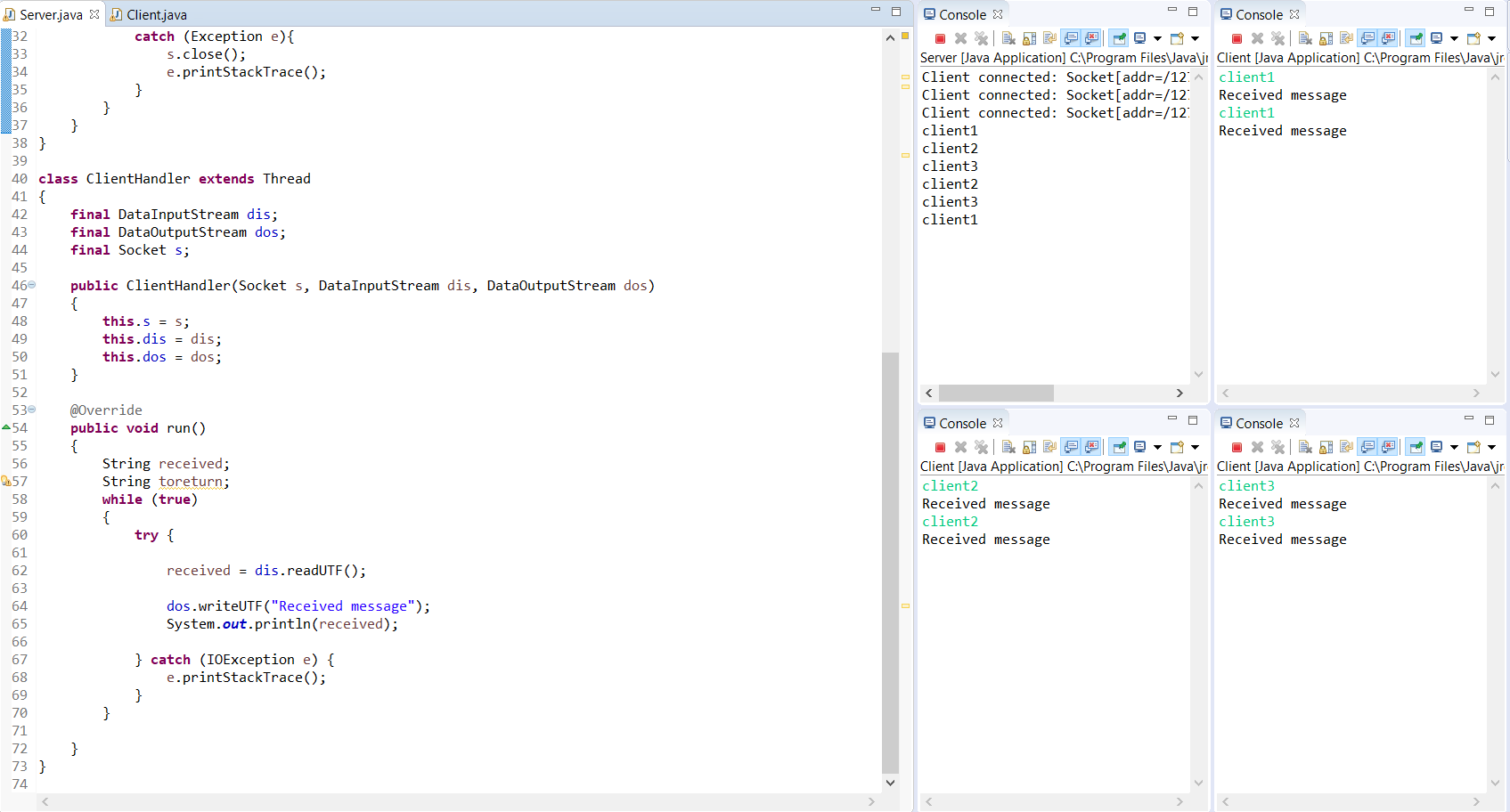
}

}

}

}





Client:

**package** endsem;

**import** java.io.\*;

**import** java.net.\*;

**import** java.util.Scanner;

**public** **class** Client

{

**public** **static** **void** main(String[] args) **throws** IOException

{

**try**

{

Scanner scn = **new** Scanner(System.***in***);

InetAddress ip = InetAddress.*getByName*("localhost");

Socket s = **new** Socket(ip, 5000);

DataInputStream dis = **new** DataInputStream(s.getInputStream());

DataOutputStream dos = **new** DataOutputStream(s.getOutputStream());

**while** (**true**)

{

String tosend = scn.nextLine();

dos.writeUTF(tosend);

System.***out***.println(dis.readUTF());

}

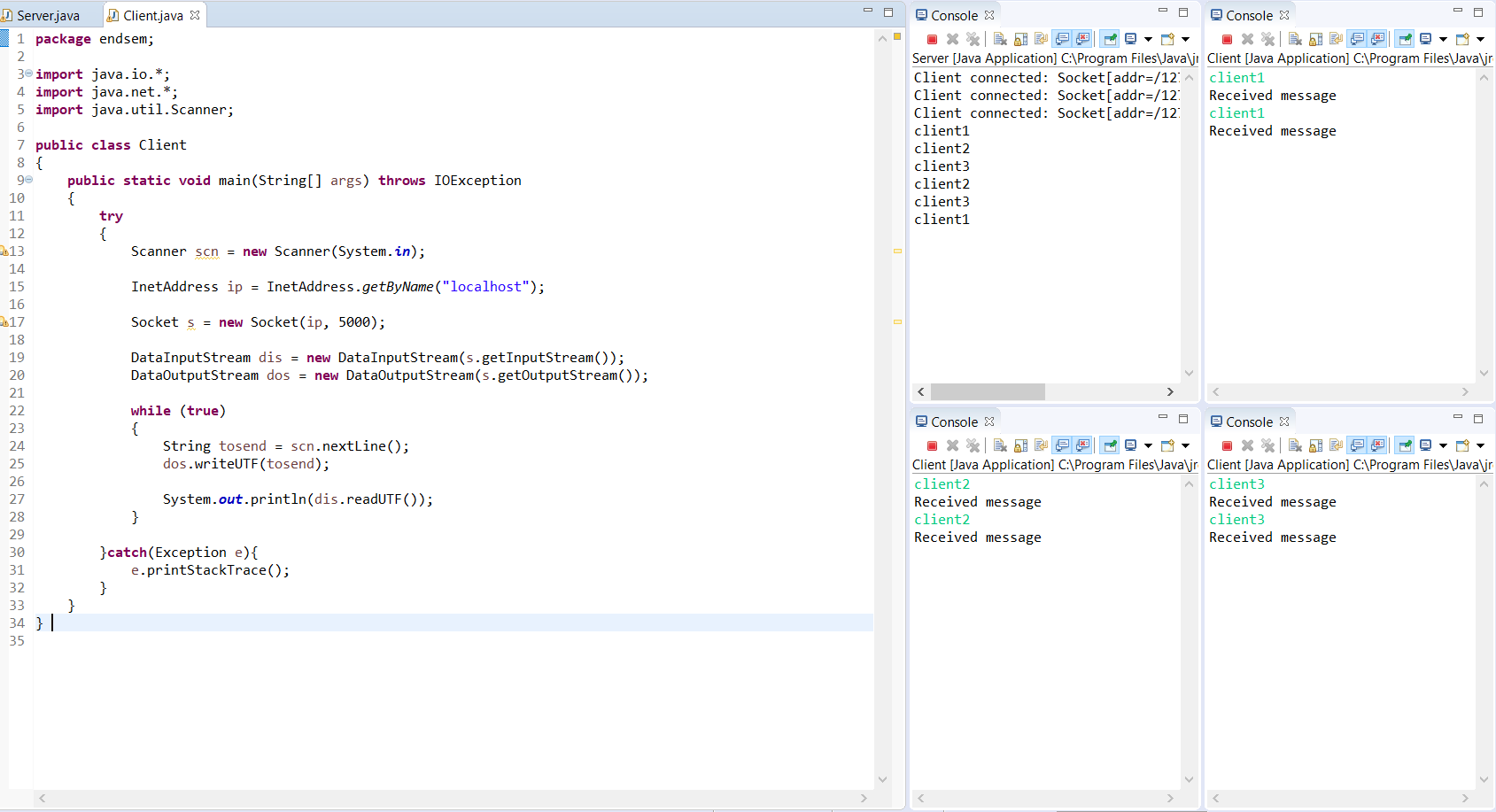
}**catch**(Exception e){

e.printStackTrace();

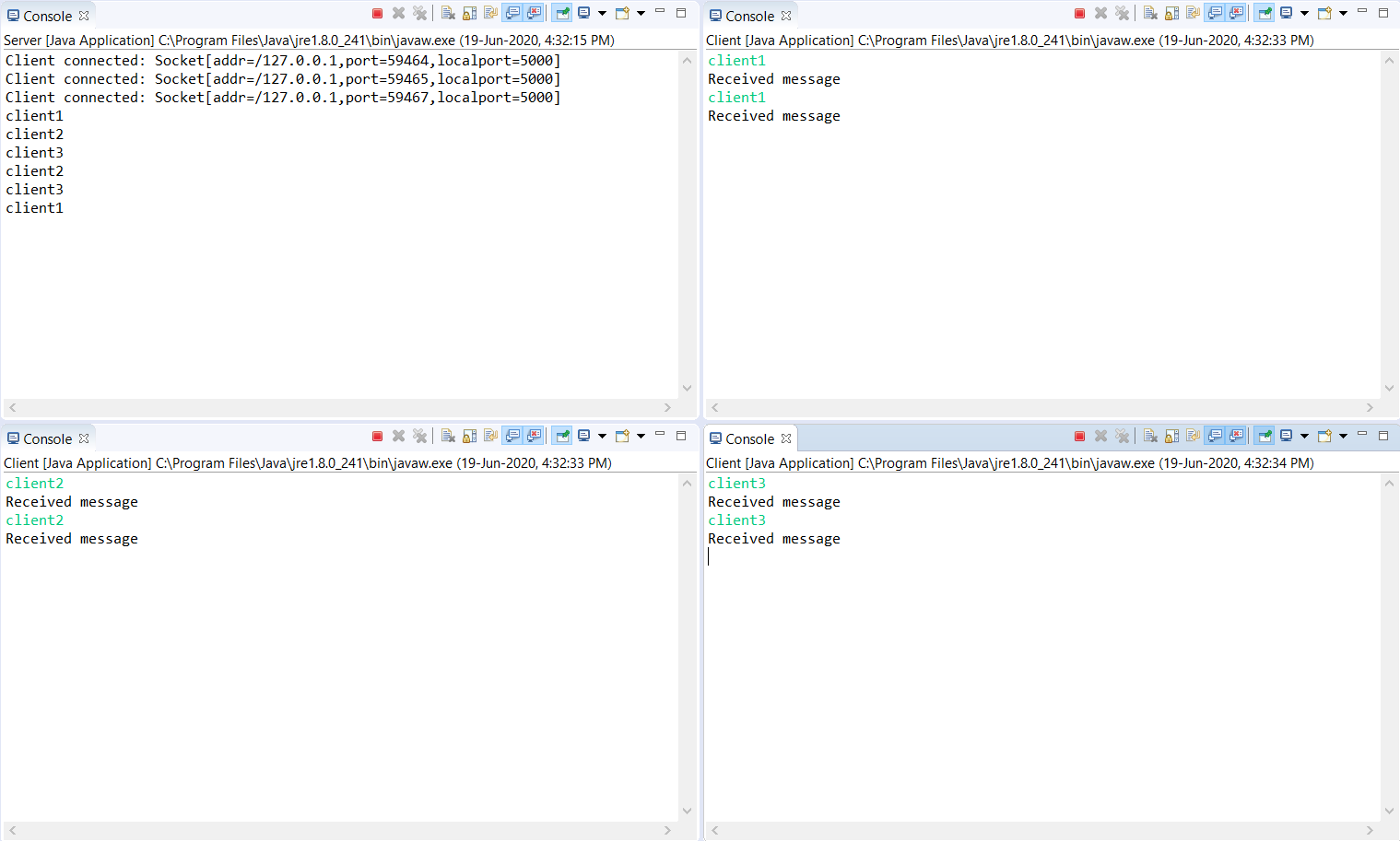
}

}

}



Output:



II. Fill in the below table for an VLSM network of class C address:192.168.10.0/24 to allocate. Cities A,B have a WAN connection to City C. The requirements for each city are

City A requires 60 hosts.

City B requires 28 hosts.

City C requires 12 hosts.

The addresses for the WAN links can be taken from the remaining usable addresses available

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Subnet | Subnet Address | Subnet Mask(/x) | First Usable Host | Last Usable Host | Broadcast |
| City A | 255.255.255.192 | /26 | 192.168.10.1 | 192.168.10.62 | 192.168.10.63 |
| City B | 255.255.255.224 | /27 | 192.168.10.65 | 192.168.10.94 | 192.168.10.95 |
| City C | 255.255.255.240 | /28 | 192.168.10.97 | 192.168.10.110 | 192.168.10.111 |
| WAN1  (A to C) | 255.255.255.252 | /30 | 192.168.10.113 | 192.168.10.114 | 192.168.10.115 |
| WAN2  (B to C) | 255.255.255.252 | /30 | 192.168.10.117 | 192.168.10.118 | 192.168.10.119 |

III. Design using packet tracer to connect any above two networks. Assign the first and last usable address to each city in the initial network.(Open the Initial network named -EndSem.pka file). Show the successful communication for

