21.Networking

- java.net package provides classes which facilitate development of networking applications in Java.
  - *Commonly used classes of this package are –*
    - InetAddress
    - Socket
    - ServerSocket
    - DatagramSocket
    - DatagramPacket
  - InetAddress class provides object representation of IP Address of machines on a network. This class doesn't provide public constructors rather it provides factory methods for creating its objects.
    - Factory is a creational design pattern that is used to control the creation of objects. This design pattern is implemented with the help of factory classes. A factory class is a class that contains factory methods.
  - A factory method is a method which creates objects.
    - *Types of Design Patterns*
- Creational Design Pattern.
- Structural Design Pattern.
- Behaviour Design Pattern.
- What is Singleton?
  - When we create only one object of a class is called singleton.
    - Example

```
class A
{
 private static final obj;

 private A()
 {}

 public static A getA()
 {
  if(obj == null)
       obj =new A();
  return obj;
 }
}
```

```
A x = new A();   ×
A x = A.getA();  ✓
A y = A.getA();  ✓
```

- *Following factory methods are provided by* **InetAddress** *class-*
  - getLocalHost() method returns an InetAddress object which represents the IP Address of the current machine.
    public Static InetAddress getLocalHost() throws UnknownHostException;

```
Example –
InetAddress a = InetAddress.getLocalHost();
```

  - getName() method returns an InetAddress object which represents IP Address of the given machine.

public static InetAddress getByName(String hostName) throws UnknownHostException;

- getAllByName() method returns an array of InetAddress objects. Each element of the array represents an IP Address of the given host.
  public static InetAddress[] getAllByName(String hostName) throws UnknownHostException;
- getHostName() method returns name of the machine.
  public String getHostName();
- getHostAddress() method returns the IP Address as a String.
  public String getHostAddress();
- etc.

```
//Program to find out IP Address of a host on a network.
//hostname is provided as command line argument.

package Networking;

import java.net.InetAddress;

public class IPFinder
{
	public static void main(String[] args)
	{
		try
		{
			InetAddress address = InetAddress.getByName(args[0]);
			System.out.println("IP Address is " + address.getHostAddress());

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

Output -

IPFinder - Notepad

```
import java.net.InetAddress;

public class IPFinder
{
        public static void main(String[] args)
        {
                try
                {
                        InetAddress address = InetAddress.getByName(args[0]);
                        System.out.println("IP Address is " + address.getHostAddress());
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

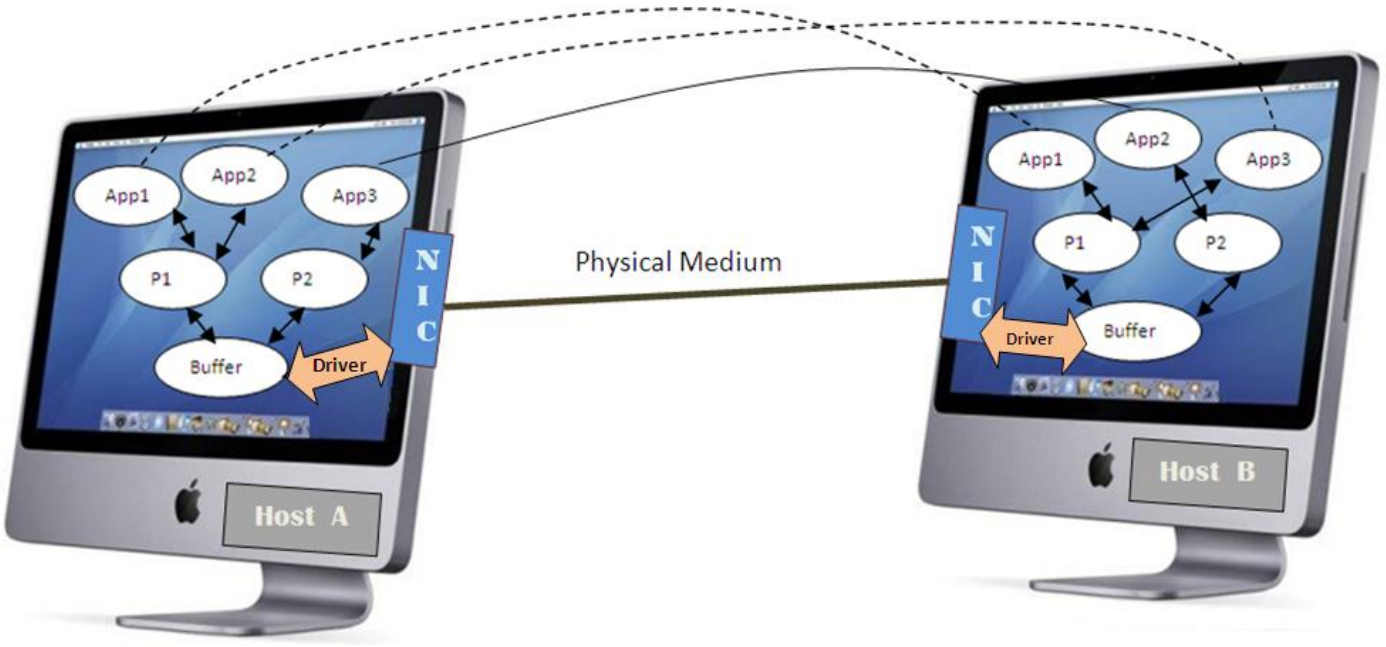
C:\WINDOWS\system32\cmd.exe

```
08/16/2010  07:16 AM    <DIR>          Program Files
08/15/2010  03:09 PM             7,996 qhdebug.log
08/01/2010  09:55 AM           539,686 QUAR.RPT
07/23/2010  10:22 AM               206 realtek.log
07/23/2010  10:22 AM               567 RHDSetup.log
05/19/2010  06:24 AM               261 Sandeep.java
07/19/2010  11:44 AM             1,679 SelectTest.class
07/19/2010  11:43 AM               866 SelectTest.java
07/19/2010  12:14 PM             1,343 SelectTest2.class
07/19/2010  12:13 PM               768 SelectTest2.java
08/22/2010  11:15 PM    <DIR>          Simple_Client_Server
08/06/2010  01:41 PM    <DIR>          temp
07/23/2010  10:26 AM    <DIR>          TempEI4
05/25/2010  01:36 PM    <DIR>          TurboC3
07/27/2010  01:20 AM    <DIR>          video_output
08/27/2010  08:56 AM    <DIR>          WINDOWS
              38 File(s)        568,174 bytes
              11 Dir(s)   5,003,501,568 bytes free

C:\>javac IPFinder.java

C:\>java IPFinder amit
IP Address is 192.168.1.2

C:\>
```



- **Socket** is a logical end point of a connection.
  - From an application programmer's point of view, a socket is a process that is used by the Application Programmer to send or retrieve data over the network.
  - This process handles protocol specific details on behalf of Application Developer.
  - To facilitate multiplexing of different logical connections over a single physical medium concept of port was introduced.
  - A port is a numbered socket.
  - Port Number 0 to 1024 are reserved for standard protocols such as TCP/IP, Http, Ftp, SMTP etc.

- java.net.Socket class represents a TCP/IP socket.

  A Socket *object can be created using either of the following constructors:-*

  public Socket (String hostname, int port) throws UnknownHostException, IOException;

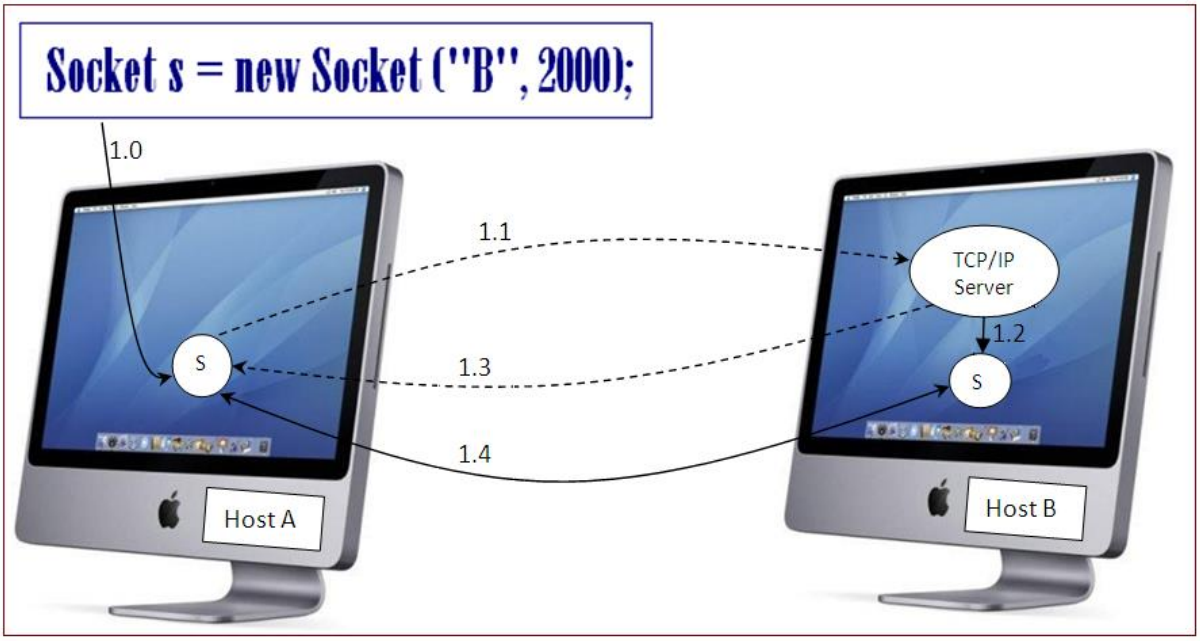
  public Socket (InetAddress ipAddress, int port) throws UnknownHostException, IOException;

- *Methods-*
- getInputStream() method returns an InputStream to read data from a Socket.

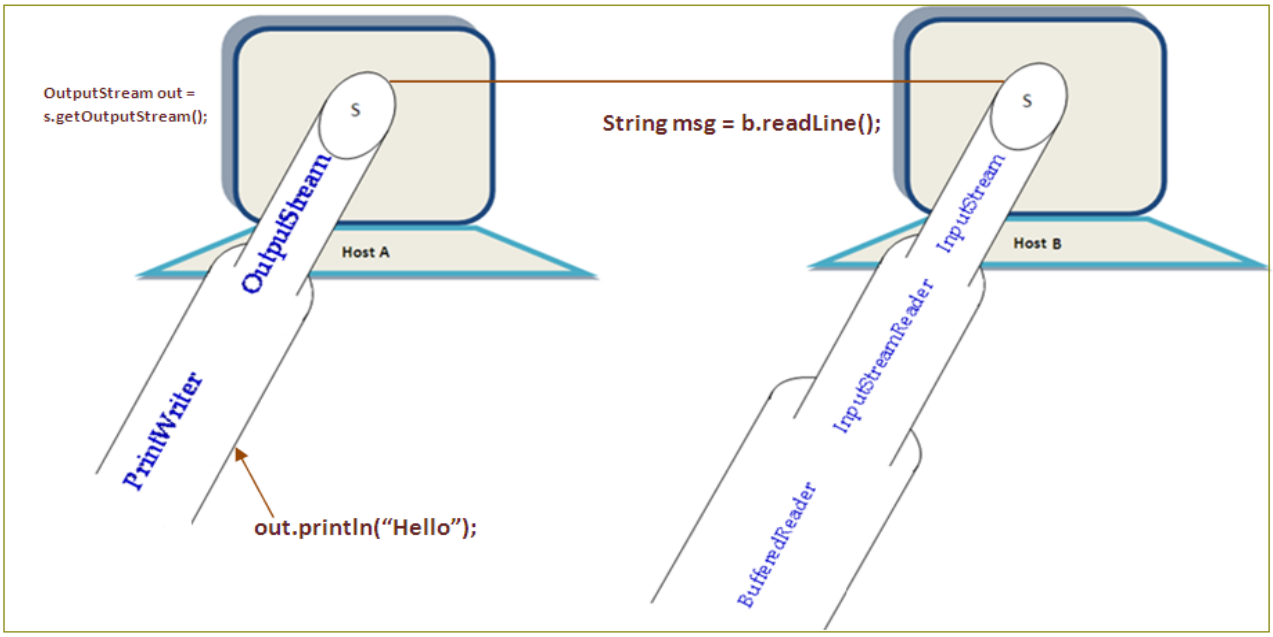
  public InputStream getInputStream();
- getOutputStream() method returns an OutputStream to write data to a Socket.

  public OutputStream getOututStream();
- close() method is used to close the Socket.

  public void close();



1.0 Socket object is created.
1.1 From the constructor of Socket, a connection request is sent to the TCP/IP Server running on specified host for a connection on given port.
1.2 If TCP/IP Server is configured for the requested port, connection is completed by creating server-side Socket.
1.3 Acknowledgement of connection is sent.
1.4 Communication is initiated.



- java.netServerSocket class represents the functionality of a TCP/IP server.
- A TCP/IP server is responsible for receiving & completing TCP/IP connection requests.
- A ServerSocket object can be created using either of the following constructor.
  public ServerSocket(int port);
  public ServerSocket(int port, int maxQueueLength);

  Methods-
  - accept() method is used to instruct TCP/IP server to start listening connection request. This method blocks the TCP/IP server until a connection request is received.
    public Socket accept();
  - close() is used to close the TCP/IP server.
    public void close();

  Program (**Server.java)**

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

class Server
{
    public static void main(String[] args)
    {
            try
            {
                    ServerSocket server = new ServerSocket(2000);
                    System.out.println("Server is ready, waiting for connection request...");
                    Socket s = server.accept();
                    System.out.println("Waiting for message...");
                    BufferedReader b = new BufferedReader(new InputStreamReader(s.getInputStream()));
```

```
                String msg = b.readLine();
                Thread.sleep(1000);
                System.out.println("Following message received : " + msg);
                System.out.println("Sending acknowledgement ....");
                Thread.sleep(2000);
                PrintWriter out = new PrintWriter(s.getOutputStream());
                out.println("Hello Client, Your message is received.");
                out.flush();
                System.out.println("Acknowledgement sent, closing connection....");
                Thread.sleep(5000);
                System.out.println("Connection closed.");
                s.close();
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
    }
}
```

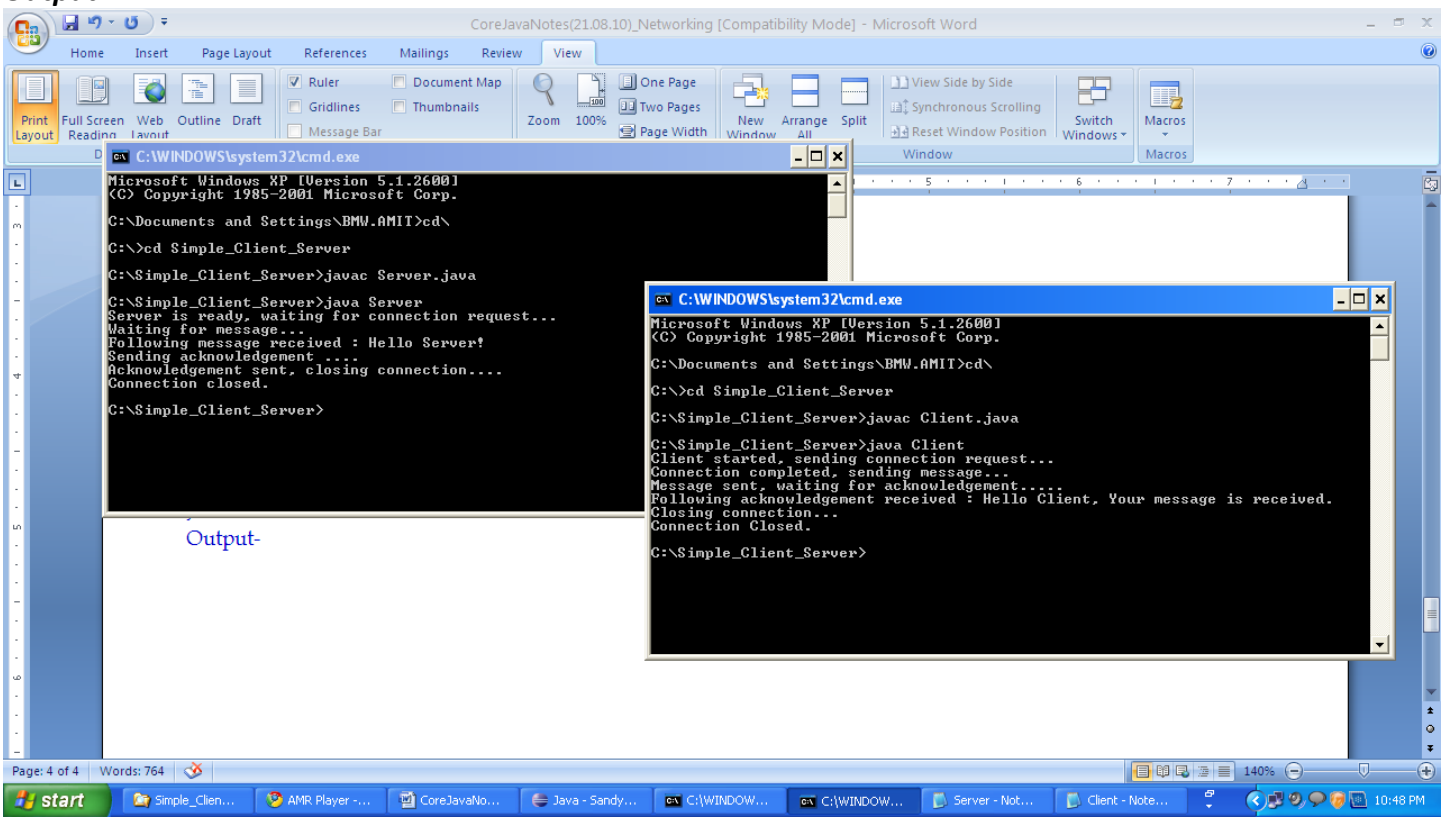
Program **(Client.java)**

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

class Client
{
    public static void main(String[] args)
    {
        try
        {
                System.out.println("Client started, sending connection request...");
                Thread.sleep(2000);
                Socket s = new Socket("localhost", 2000);
                Thread.sleep(1000);
                System.out.println("Connection completed, sending message...");
                PrintWriter out = new PrintWriter(s.getOutputStream());
                Thread.sleep(2000);
                out.println("Hello Server!");
                out.flush();
                System.out.println("Message sent, waiting for acknowledgement.....");
                BufferedReader b = new BufferedReader(new InputStreamReader(s.getInputStream()));
                String msg = b.readLine();
                Thread.sleep(1000);
                System.out.println("Following acknowledgement received : " + msg);
                System.out.println("Closing connection...");
                Thread.sleep(5000);
```

```
                System.out.println("Connection Closed.");
                s.close();
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
    }
}
```

Output-



- DatagramSocket class provides the facility of sending & receiving UDP packets.
  public DatagramSocket (int port);
  Methods-
  - send() method is used to send UDP packets.
    public void send(DatagramPacket packet);
  - receive() method is used to receive UDP packets.
    public void receive(DatagramPacket packet);
  - close() method is used to close DatagramSocket.
    public void close();
- DatagramPacket class provides the object representation of UDP packets.

❖ *Constructors are –*

public DatagramPacket (byte[] data, int size, InetAddress hostAddress, int port); // used to send data
public DatagramPacket (byte[] data, int size); // used to receive data

❖ *Methods are –*

public byte[] getData();
public InetAddress getHost();
public int getPort();
etc.

Program (**UdpSender.java)**

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

class UdpSender
{
    public static void main(String[] args)
    {
            try
            {
                    DatagramSocket sender = new DatagramSocket(3000);
                    Scanner in = new Scanner(System.in);
                    while(true)
                    {
                            System.out.println("Enter Message, end to terminate...");
                            String msg = in.nextLine();
                            if(msg.equals("end"))
                                    break;
                            DatagramPacket packet = new DatagramPacket(msg.getBytes(), msg.length(),
InetAddress.getLocalHost(), 4000);
                            sender.send(packet);
                            System.out.println("Successfully sent.");
                    }
                    sender.close();
            }
            catch(Exception e)
            {
                    System.out.println(e);
            }
    }
}
```

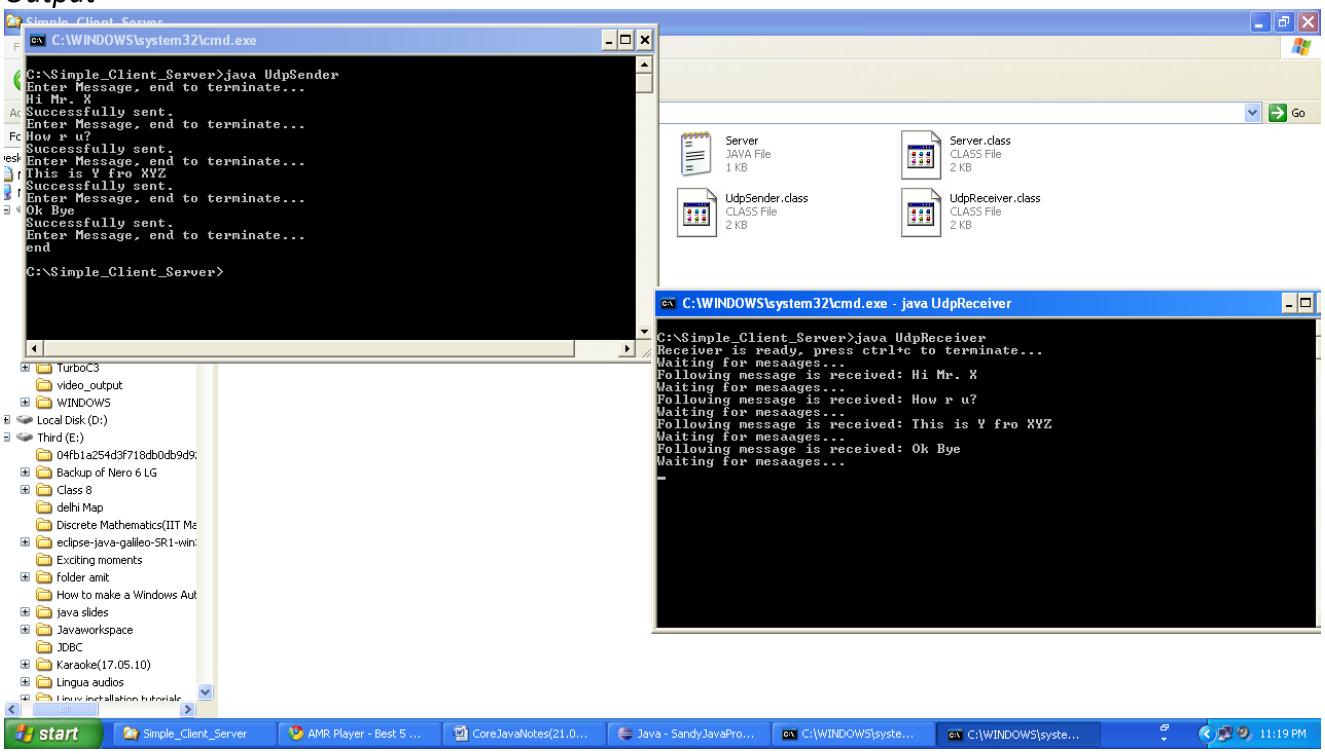
Program (**UdpReceiver.java)**

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

class UdpReceiver
```

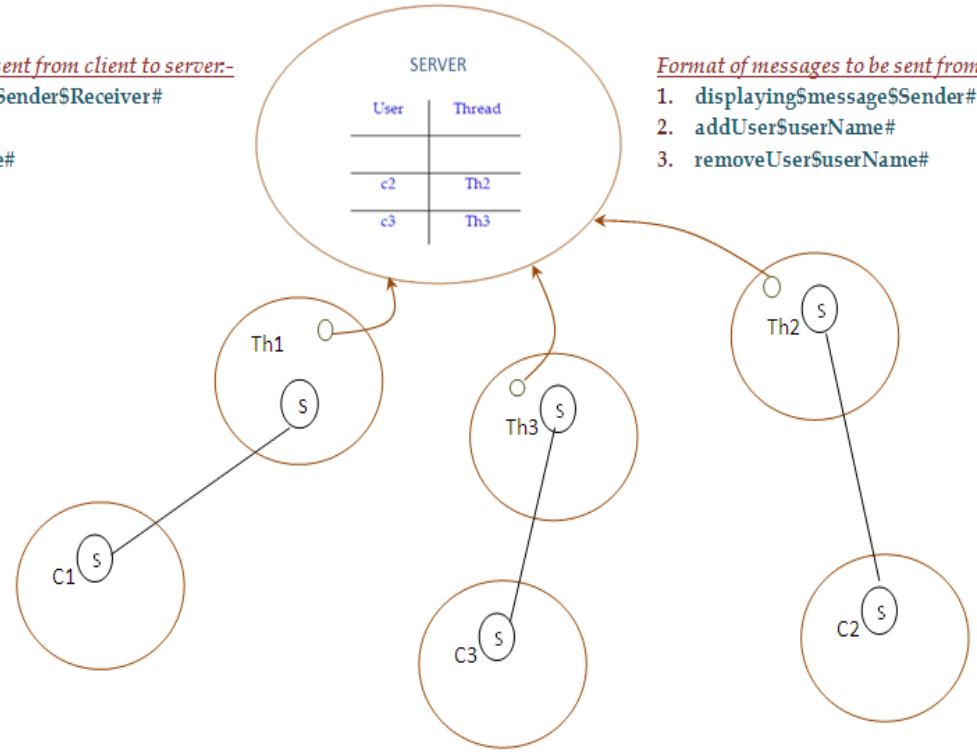
```
{
    public static void main(String[] args)
    {
            try
            {
                    DatagramSocket receiver = new DatagramSocket(4000);
                    System.out.println("Receiver is ready, press ctrl+c to terminate...");
                    while(true)
                    {
                            System.out.println("Waiting for mesaages...");
                            DatagramPacket packet = new DatagramPacket(new byte[100], 100);
                            receiver.receive(packet);
                            String msg = new String(packet.getData());
                            System.out.println("Following message is received: " + msg.trim());
                    }
            }
            catch(Exception e)
            {
                    System.out.println(e);
            }
    }
}
```

Output-



Format of messages to be sent from client to server:-

1. delivering$message$Sender$Receiver#
2. Connect$userName#
3. disconnect$userName#



Format of messages to be sent from server to client:-

1. displaying$message$Sender#
2. addUser$userName#
3. removeUser$userName#