LAB 01

**Lab1task1.java**

import java.net.\*;

import java.io.\*;

public class Lab1Task1{

public static void main(String[] args){

// this application sends message using connectionless datagram socket

if(args.length!=6)

System.out.println("this program requires six command line arguments");

else{

try{

InetAddress receiverHost=InetAddress.getByName(args [0]);

int receiverPort= Integer.parseInt(args [1]);

String message=args[2];

DatagramSocket mySocket=new DatagramSocket();

byte[] buffer=message.getBytes();

DatagramPacket datagram=new DatagramPacket(buffer,buffer.length,receiverHost,receiverPort);

mySocket.send(datagram);

mySocket.close();

}

catch(Exception e){

e.printStackTrace();

}

try{

InetAddress receiverHost1=InetAddress.getByName(args[3]);

int receiverPort1= Integer.parseInt(args [4]);

String mess1=args[5];

DatagramSocket mySocket1=new DatagramSocket();

byte[] buffer1=mess1.getBytes();

DatagramPacket datagram1=new DatagramPacket(buffer1,buffer1.length,receiverHost1,receiverPort1);

mySocket1.send(datagram1);

mySocket1.close();

}

catch(Exception e){

e.printStackTrace(); }} } }

**ExampleReceiver.java**

import java.net.\*;

import java.io.\*;

public class ExampleReceiver{

public static void main(String[] args){

if (args.length!=1)

System.out.println("This program requires a command line argument.");

else{

int port =Integer.parseInt(args[0]);

final int MAX\_LEN=10;

try{

DatagramSocket mySocket= new DatagramSocket(port);

byte[] buffer= new byte[MAX\_LEN];

DatagramPacket datagram= new DatagramPacket(buffer, MAX\_LEN);

mySocket.receive(datagram);

String message= new String(buffer);

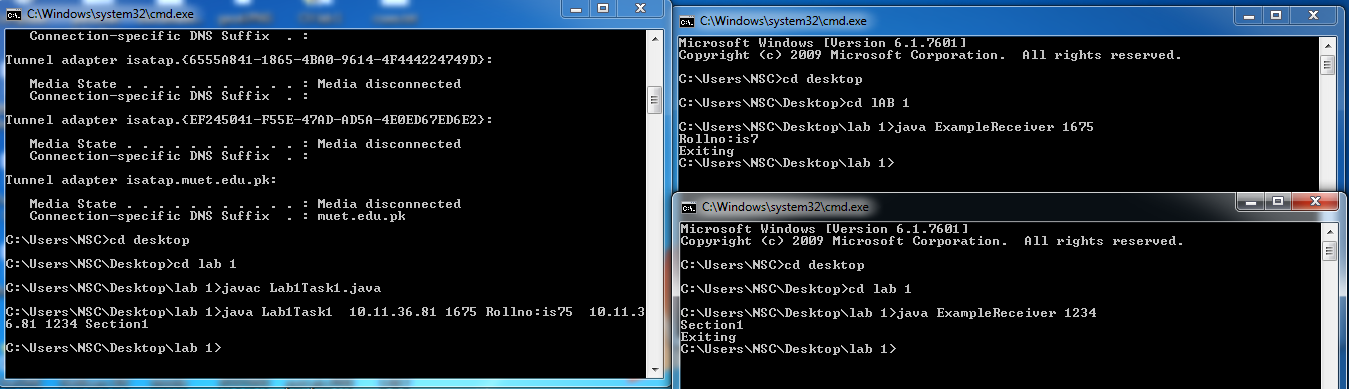
System.out.println(message);

Thread.sleep(30000);

System.out.print("Exiting");

mySocket.close(); }

catch(Exception ex)

{ ex.printStackTrace();} } } }

**Lab1Task2.java**

import java.net.\*;

import java.io.\*;

public class Lab1Task2{

public static void main(String[] args){

// this application sends message using connectionless datagram socket

if(args.length!=3)

System.out.println("requires 3 command line arguments");

else{

try{

InetAddress receiverHost=InetAddress.getByName(args [0]);

int receiverPort= Integer.parseInt(args [1]);

String message=args[2];

DatagramSocket mySocket=new DatagramSocket();

byte[] buffer=message.getBytes();

DatagramPacket datagram=new DatagramPacket(buffer,buffer.length,receiverHost,receiverPort);

mySocket.send(datagram);

mySocket.close();}

catch(Exception e){

e.printStackTrace(); }}}}

**ServerTask2.java**

import java.net.\*;

import java.io.\*;

public class ServerTask2{

public static void main(String[] args){

int a=0;

if (args.length!=1)

System.out.println("This program requires a command line argument.");

else{

int port =Integer.parseInt(args[0]);

while(a<5){

final int MAX\_LEN=10; try{

DatagramSocket mySocket= new DatagramSocket(port);

byte[] buffer= new byte[MAX\_LEN];

DatagramPacket datagram= new DatagramPacket(buffer, MAX\_LEN);

mySocket.receive(datagram);

String message= new String(buffer);

System.out.println(message);

Thread.sleep(10000);

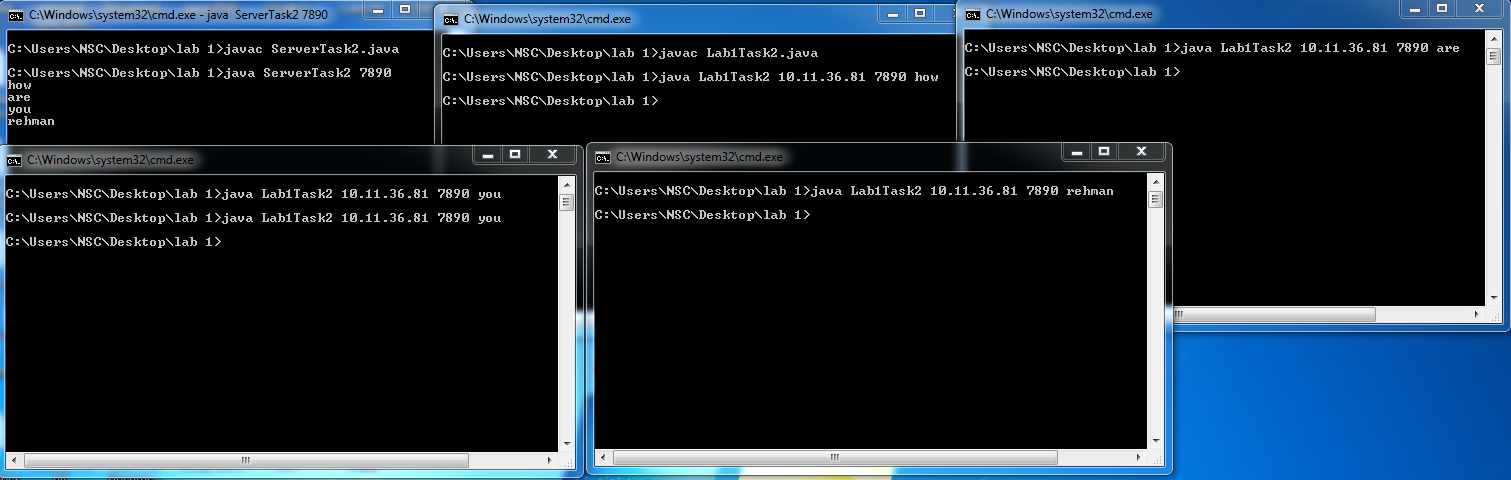
mySocket.close(); }

catch(Exception ex)

{ ex.printStackTrace();

a++;}

} } } }



**ChatServer.java**

import java.io.\*;

import java.net.\*;

public class ChatServer{

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

ServerSocket sersock = new ServerSocket(3000);

System.out.println("Server ready for chatting");

Socket sock = sersock.accept( );

// reading from keyboard (keyRead object)

BufferedReader kRead = new BufferedReader(new InputStreamReader(System.in));

// sending to client (pwrite object)

OutputStream ostream = sock.getOutputStream();

PrintWriter p = new PrintWriter(ostream, true);

// receiving from server ( receiveRead object)

InputStream istream = sock.getInputStream();

BufferedReader reader = new BufferedReader(new InputStreamReader(istream));

String receiveMessage, sendMessage;

System.out.println("server :");

while(true) {

if((receiveMessage = reader.readLine()) != null) {

System.out.println(receiveMessage); }

sendMessage = kRead.readLine();

p.println(sendMessage);

p.flush();} }}

**ChatClient.java**

import java.io.\*;

import java.net.\*;

public class ChatServer{

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

ServerSocket sersock = new ServerSocket(3000);

System.out.println("Server ready for chatting");

Socket sock = sersock.accept( );

// reading from keyboard (keyRead object)

BufferedReader kRead = new BufferedReader(new InputStreamReader(System.in));

// sending to client (pwrite object)

OutputStream ostream = sock.getOutputStream();

PrintWriter p = new PrintWriter(ostream, true);

// receiving from server ( receiveRead object)

InputStream istream = sock.getInputStream();

BufferedReader reader = new BufferedReader(new InputStreamReader(istream));

String receiveMessage, sendMessage;

System.out.println("server :");

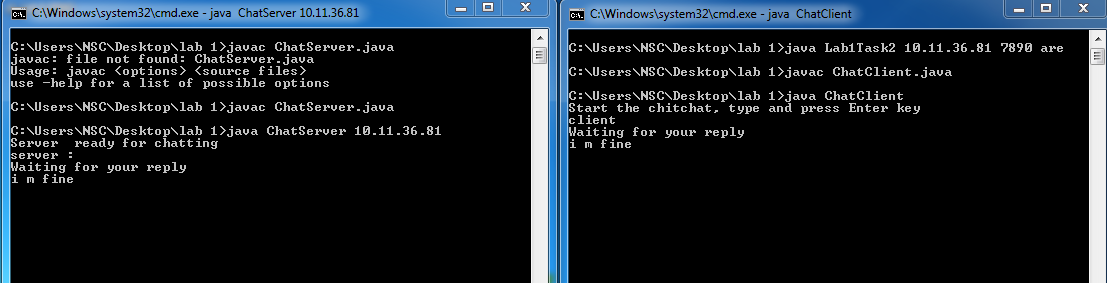
while(true) {

if((receiveMessage = reader.readLine()) != null)

{System.out.println(receiveMessage); }

sendMessage = kRead.readLine();

p.println(sendMessage);

 p.flush(); }}}

**Task4Receiver.java**

import java.net.\*;

import java.io.\*;

public class Task4Receiver{

public static void main(String[] args){

if (args.length!=1)

System.out.println("This program requires a command line argument.");

else{

int port =Integer.parseInt(args[0]);

final int MAX\_LEN=10;

try{

DatagramSocket mySocket= new DatagramSocket(port);

byte[] buffer= new byte[MAX\_LEN];

DatagramPacket datagram= new DatagramPacket(buffer, MAX\_LEN);

mySocket.receive(datagram);

String message= new String(buffer);

System.out.println(message);

Thread.sleep(30000);

mySocket.close(); }

catch(Exception ex)

{ ex.printStackTrace();} } } }

**BroadcastingClientTask1.java**

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class BroadcastingClientTask1 {

static DatagramSocket socket = null;

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

broadcast("AbdulRehman", InetAddress.getByName("255.255.255.255"));}

public static void broadcast(

String broadcastMessage, InetAddress address) throws IOException {

socket = new DatagramSocket();

socket.setBroadcast(true);

byte[] buffer = broadcastMessage.getBytes();

DatagramPacket packet

= new DatagramPacket(buffer, buffer.length, address, 1001);

socket.send(packet);

socket.close(); }

List<InetAddress> listAllBroadcastAddresses() throws SocketException {

List<InetAddress> broadcastList = new ArrayList<>();

Enumeration<NetworkInterface> interfaces = NetworkInterface.getNetworkInterfaces();

while (interfaces.hasMoreElements()) {

NetworkInterface networkInterface = interfaces.nextElement();

if (networkInterface.isLoopback() || !networkInterface.isUp()) {

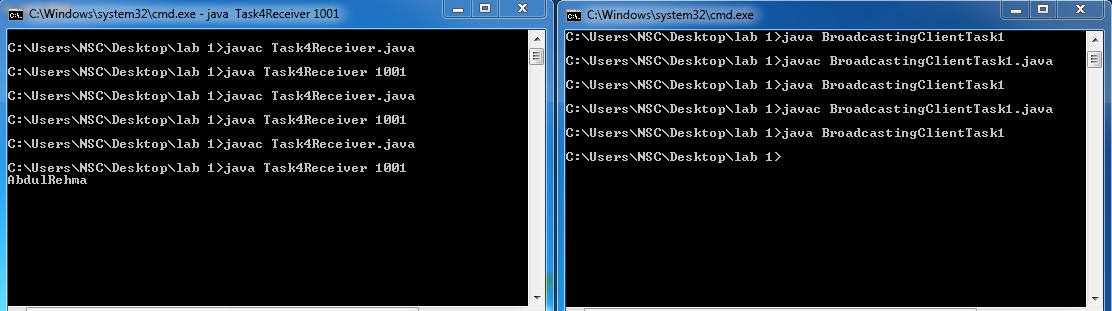
continue; }

networkInterface.getInterfaceAddresses().stream()

.map(a -> a.getBroadcast())

.filter(Objects::nonNull)

.forEach(broadcastList::add); }

 return broadcastList;}}

**MulticastPublisherTask5.java**

import java.net.\*;

import java.io.\*;

public class MulticastPublisherTask5 {

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

multicast("16SW75");}

private static DatagramSocket socket;

private static InetAddress group;

private static byte[] buf;

public static void multicast(String multicastMessage) throws IOException {

socket = new DatagramSocket();

group = InetAddress.getByName("230.0.0.0");

buf = multicastMessage.getBytes();

DatagramPacket packet = new DatagramPacket(buf, buf.length, group, 4446);

socket.send(packet);

socket.close();}}

**MulticastReceiverTask5.java**

import java.net.\*;

import java.io.\*;

public class MulticastReceiverTask5 {

protected static MulticastSocket socket = null;

protected static byte[] buf = new byte[256];

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

socket = new MulticastSocket(4446);

InetAddress group = InetAddress.getByName("230.0.0.0");

socket.joinGroup(group);

while (true) {

DatagramPacket packet = new DatagramPacket(buf, buf.length);

socket.receive(packet);

String message=new String(buf);

System.out.println(message);

String received = new String(packet.getData(), 0, packet.getLength());

if ("end".equals(received)) {

break;}}

socket.leaveGroup(group);

socket.close();}}

