**LAB # 1 :-**

**OBJECT : To learn creation of datagram sockets.**

**Lab Tasks:**

1. **Modify the sample code so that the sender uses the same socket to send the same message to two different receivers. Start the two receivers first, then the sender. Does each receiver receive the message? Capture the code and output. Describe the outcome.**

**Program:**

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(10000);

System.out.print("Exiting");

mySocket.close(); }

catch(Exception ex)

{ ex.printStackTrace();}

}

} }

import java.net.\*;

import java.io.\*;

public class ExampleReceiver2{

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(10000);

System.out.print("Exiting");

mySocket.close(); }

catch(Exception ex)

{ ex.printStackTrace();}

}

} }

import java.net.\*;

import java.io.\*;

public class ExampleSender {

public static void main(String[] args){

// this application sends message using connectionless datagram socket

if(args.length!=4)

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

else{

try{

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

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

int port2 = Integer.parseInt(args [2]);

String message=args[3];

DatagramSocket mySocket=new DatagramSocket();

byte[] buffer=message.getBytes();

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

mySocket.send(datagram);

mySocket.close();

DatagramSocket mySocket1=new DatagramSocket();

DatagramPacket datagram1=new DatagramPacket(buffer,buffer.length,receiverHost,port2);

mySocket1.send(datagram1);

mySocket1.close();

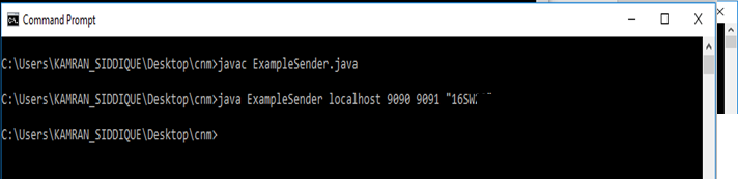
}

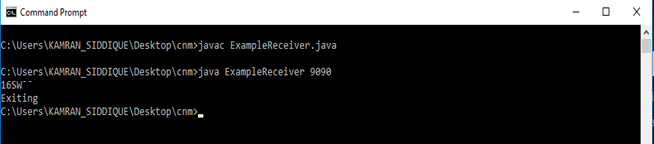
catch(Exception e){

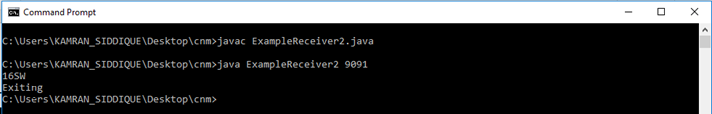
e.printStackTrace();

}

} } }







**2. Modify the sample code so that the receiver loops five times to repeatedly receive then display your bio data (name, roll num etc.) received. Recompile. Then**

**i. start the receiver**

**ii. Execute the sender, sending your bio data, and**

**iii. In another window, start another instance of the sender, sending your friend’s bio data. Does the receiver receive both the messages? Capture the code and output.**

**Program :**

import java.net.\*;

import java.io.\*;

public class TASK2Sender {

public static void main(String[] args){

// this application sends message using connectionless datagram socket

if(args.length!=3)

System.out.println("this program requires three 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();

}

} } }

import java.net.\*;

import java.io.\*;

public class TASK2Receiver{

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

{

String msg[] =new String [5];

int i = 0;

while(i<5)

{

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(1000);

mySocket.close();

msg[i++] = message;

System.out.println(i+" message arrived.");

}

System.out.println("\nFive messages are : ");

for (int j = 0;j<5;j++)

{

System.out.println(msg[j]);

}

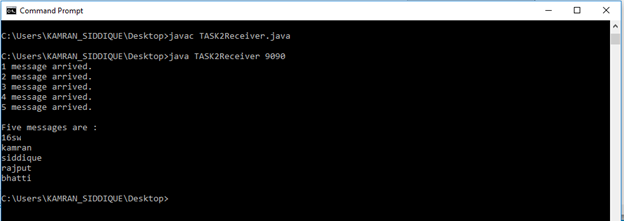
}

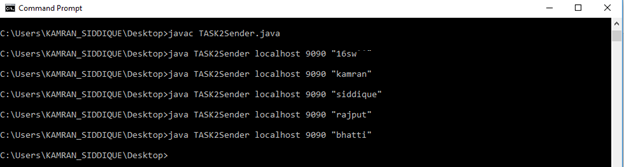
catch(Exception ex)

{ ex.printStackTrace();}

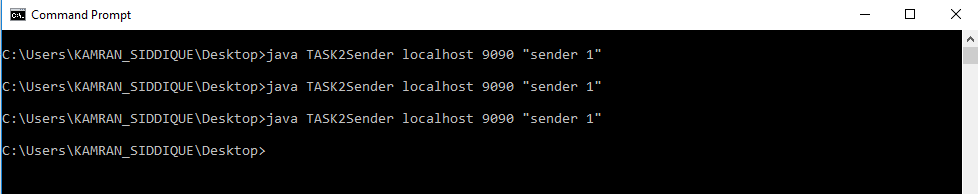
}

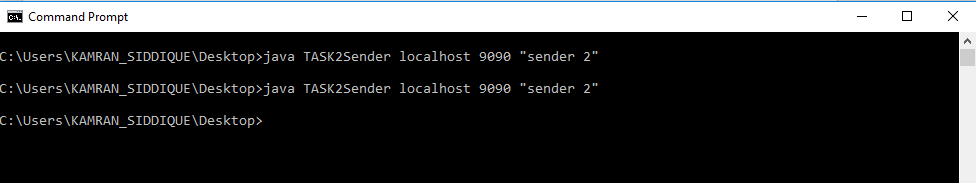
} }

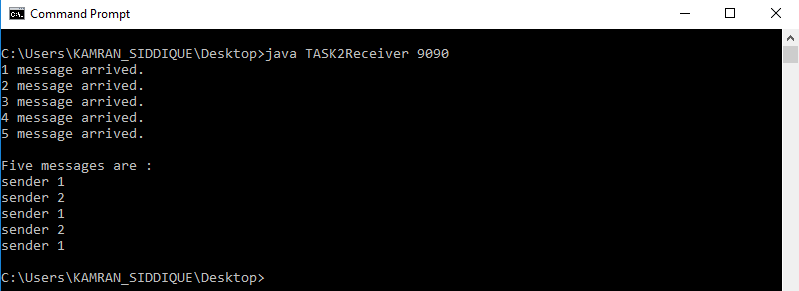


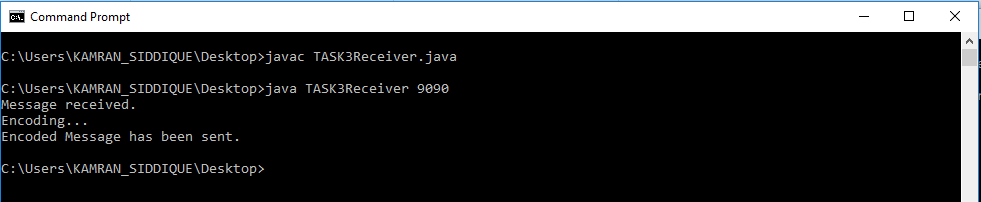


Multiple sender









**3-Modify the sample code to cater to a two way communication i.e. Sender sends a message to the Receiver, the Receiver receives the message and sends a reply to the Sender in return.**

Client.java:

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.OutputStreamWriter;

import java.net.InetAddress;

import java.net.Socket;

public class Client

{

private static Socket socket;

public static void main(String args[])

{

try

{

String host = "localhost";

int port = 25000;

InetAddress address = InetAddress.getByName(host);

socket = new Socket(address, port);

//Send the message to the server

OutputStream os = socket.getOutputStream();

OutputStreamWriter osw = new OutputStreamWriter(os);

BufferedWriter bw = new BufferedWriter(osw);

String number = "2";

String sendMessage = number + "\n";

bw.write(sendMessage);

bw.flush();

System.out.println("Message sent to the server : "+sendMessage);

//Get the return message from the server

InputStream is = socket.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

String message = br.readLine();

System.out.println("Message received from the server : " +message);

}

catch (Exception exception)

{

exception.printStackTrace();

}

finally

{

//Closing the socket

try

{

socket.close();

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

}

Server.java:

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.OutputStreamWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class Server

{

private static Socket socket;

public static void main(String[] args)

{

try

{

int port = 25000;

ServerSocket serverSocket = new ServerSocket(port);

System.out.println("Server Started and listening to the port 25000");

//Server is running always. This is done using this while(true) loop

while(true)

{

//Reading the message from the client

socket = serverSocket.accept();

InputStream is = socket.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

String number = br.readLine();

System.out.println("Message received from client is "+number);

//Multiplying the number by 2 and forming the return message

String returnMessage;

try

{

int numberInIntFormat = Integer.parseInt(number);

int returnValue = numberInIntFormat\*2;

returnMessage = String.valueOf(returnValue) + "\n";

}

catch(NumberFormatException e)

{

//Input was not a number. Sending proper message back to client.

returnMessage = "Please send a proper number\n";

}

//Sending the response back to the client.

OutputStream os = socket.getOutputStream();

OutputStreamWriter osw = new OutputStreamWriter(os);

BufferedWriter bw = new BufferedWriter(osw);

bw.write(returnMessage);

System.out.println("Message sent to the client is "+returnMessage);

bw.flush();

}

}

catch (Exception e)

{

e.printStackTrace();

}

finally

{

try

{

socket.close();

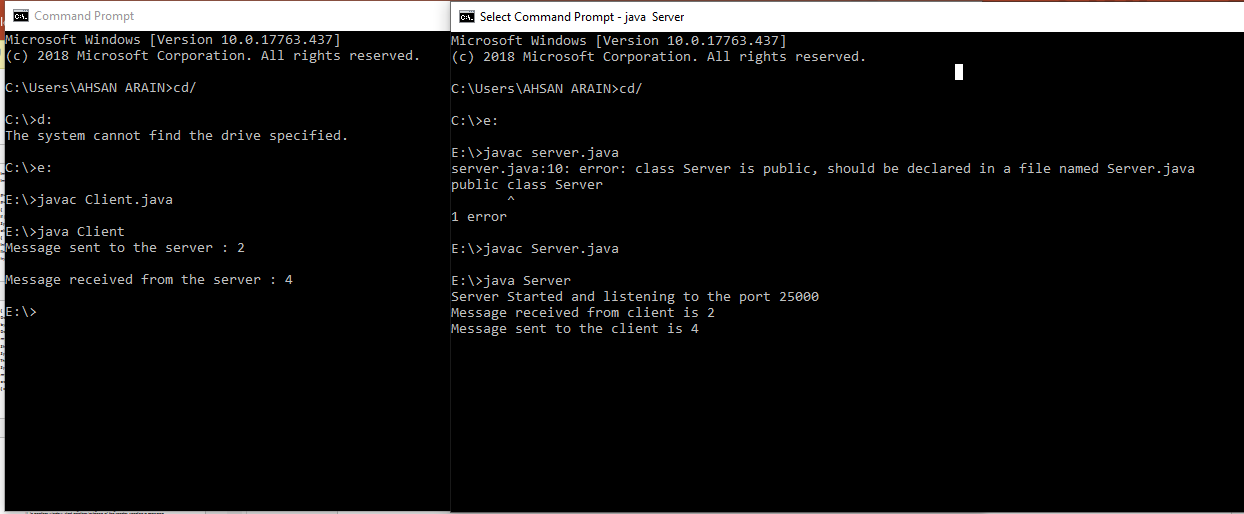
}

catch(Exception e){}

}

}

}



**4-Broadcasting: Broadcasting is a one-to-all type of communication, i.e. the intention is to send the datagram to all the nodes in the network. Unlike in the case of point-to-point communication, we don’t have to know the target host’s IP Address. Instead, a broadcast address is used.**

**Implement two simple programs using Java datagram sockets, which broadcasts and multicast your roll number to all or selected network nodes respectively.**

BroadcastClient.java:

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class BroadcastClient {

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

MulticastSocket socket = new MulticastSocket(4446);

InetAddress address = InetAddress.getByName("230.0.0.1");

socket.joinGroup(address);

DatagramPacket packet;

// get roll no

byte[] buf = new byte[256];

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

socket.receive(packet);

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

System.out.println("your roll no is: " + received);

socket.leaveGroup(address);

socket.close();

}

}

BroadcastServer.java:

public class Broadcastserver {

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

new BroadcastServerThread().start();

}

}

BroadcastServerThread.java:

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class BroadcastServerThread extends QuoteServerThread{

private long FIVE\_SECONDS = 5000;

public BroadcastServerThread() throws IOException {

super("MulticastServerThread");

}

public void run() {

try {

byte[] buf = new byte[256];

String dString = null;

if (in == null)

dString = new Date().toString();

else

dString = getNextQuote();

buf = dString.getBytes();

// send it

InetAddress group = InetAddress.getByName("230.0.0.1");

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

socket.send(packet);

// sleep for a while

try {

sleep((long)(Math.random() \* FIVE\_SECONDS));

} catch (InterruptedException e) { }

} catch (IOException e) {

e.printStackTrace();

}

socket.close();

}

}

QuoteServer.java:

import java.io.\*;

public class QuoteServer {

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

new QuoteServerThread().start();

}

}

QuoteServerThread.java:

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class QuoteServerThread extends Thread {

protected DatagramSocket socket = null;

protected BufferedReader in = null;

protected boolean moreQuotes = true;

public QuoteServerThread() throws IOException {

this("QuoteServerThread");

}

public QuoteServerThread(String name) throws IOException {

super(name);

socket = new DatagramSocket(4445);

try {

in = new BufferedReader(new FileReader("one-liners.txt"));

} catch (FileNotFoundException e) {

System.err.println("Could not open quote file. Serving time instead.");

}

}

public void run() {

try {

byte[] buf = new byte[256];

// receive request

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

socket.receive(packet);

// figure out response

String dString = null;

if (in == null)

dString = new Date().toString();

else

dString = getNextQuote();

buf = dString.getBytes();

// send the response to the client at "address" and "port"

InetAddress address = packet.getAddress();

int port = packet.getPort();

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

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

socket.close();

}

protected String getNextQuote() {

String returnValue = null;

try {

if ((returnValue = in.readLine()) == null) {

in.close();

moreQuotes = false;

returnValue = "end";

}

} catch (IOException e) {

returnValue = "IOException occurred in server.";

}

return returnValue;

}

}

5-Multicasting: Broadcasting is inefficient as packets are sent to all nodes in the network, irrespective of whether they are interested in receiving the communication or not. This may be a waste of resources. Multicasting sends packets to only those nodes which are interested. Multicasting is based on a group membership concept, where a multicast address represents each group.

UDPMulticastClient.java:

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.InetAddress;

import java.net.MulticastSocket;

public class UDPMulticastClient implements Runnable {

public static void main(String[] args) {

Thread t=new Thread(new UDPMulticastClient());

t.start();

}

public void receiveUDPMessage(String ip, int port) throws

IOException {

byte[] buffer=new byte[1024];

MulticastSocket socket=new MulticastSocket(4321);

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

socket.joinGroup(group);

while(true){

System.out.println("Waiting for multicast message...");

DatagramPacket packet=new DatagramPacket(buffer,

buffer.length);

socket.receive(packet);

String msg=new String(packet.getData(),

packet.getOffset(),packet.getLength());

System.out.println("[Multicast UDP message received]>>"+msg);

if("OK".equals(msg)) {

System.out.println("No more message. Exiting : "+msg);

break;

}

}

socket.leaveGroup(group);

socket.close();

}

@Override

public void run(){

try {

receiveUDPMessage("230.0.0.0", 4321);

}catch(IOException ex){

ex.printStackTrace();

}

}

}

UDPMulticastServer.java:

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

public class UDPMulticastServer {

public static void sendUDPMessage(String message,

String ipAddress, int port) throws IOException {

DatagramSocket socket = new DatagramSocket();

InetAddress group = InetAddress.getByName(ipAddress);

byte[] msg = message.getBytes();

DatagramPacket packet = new DatagramPacket(msg, msg.length,

group, port);

socket.send(packet);

socket.close();

}

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

sendUDPMessage("This is a multicast messge", "230.0.0.0",

4321);

sendUDPMessage("This is the second multicast messge",

"230.0.0.0", 4321);

sendUDPMessage("This is the third multicast messge",

"230.0.0.0", 4321);

sendUDPMessage("OK", "230.0.0.0", 4321);

}

}