Department of Software Engineering Mehran University of Engineering and Technology, Jamshoro

Course: SW422 – Distributed Computing			
Instructor	Rabeea Jaffari	**Practical/Lab No.**	01
Date		**CLOs**	CLO-3: P5 & CLO-3: P3
Signature		**Assessment Score**	

Topic	**To work with Socket Programming API for client-server distributed application**
Objectives	- Learn Socket API basics using Datagram Sockets

Lab Discussion: Theoretical concepts and Procedural steps

Lab Tasks

Submission Date: 22 April, 2019.

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.

Receiver1.java | Sender.java | Client.java | Server.java

```java
import java.net.*;
import java.io.*;
public class Sender {
    public static void main(String[] args){
          // this application sends message using connectionless datagram socket
          // 16SW04
          // Jawaria Sattar
     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();
         }
     }  } }
```

Receiver.java* | Sender.java | Client.java

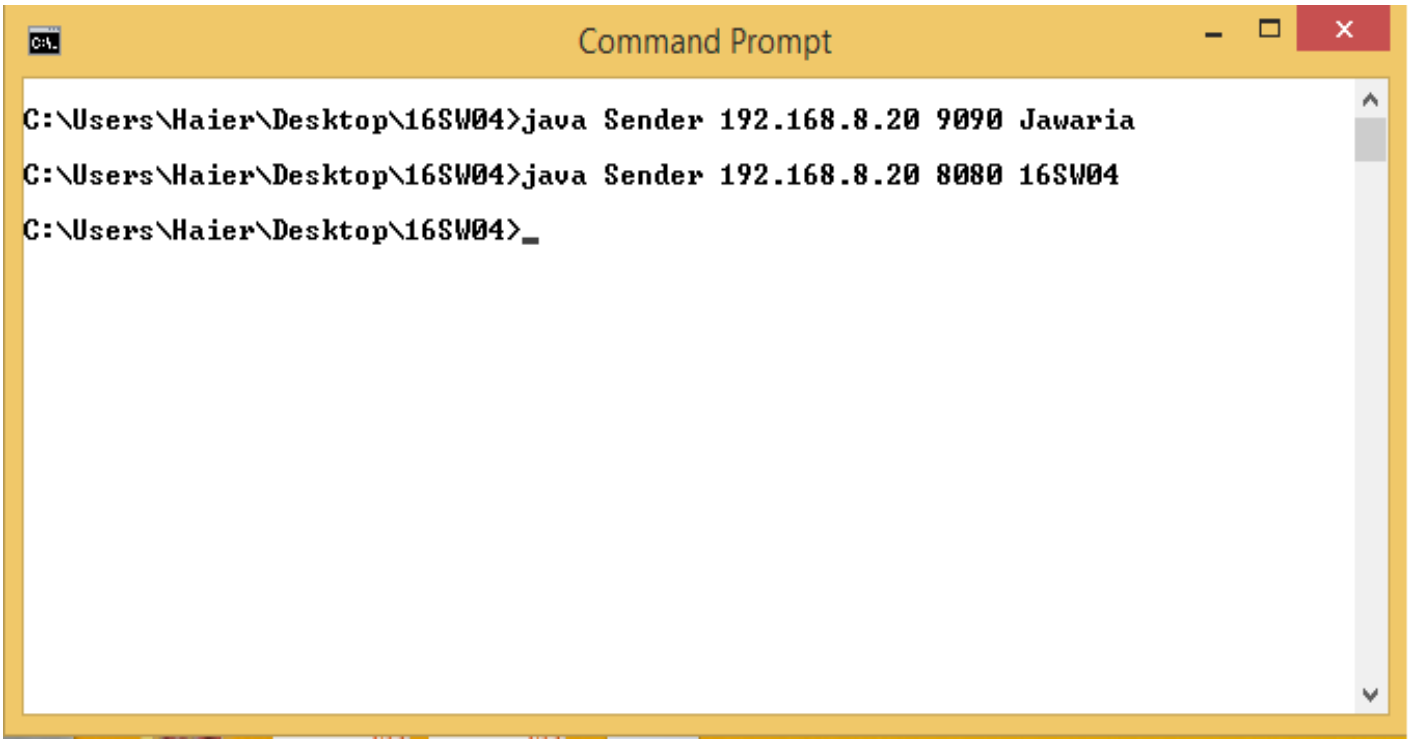
```java
import java.net.*;
import java.io.*;

public class Receiver{
public static void main(String[] args)
{
        // 16SW04
            // Jawaria Sattar

if (args.length!=1)
System.out.println("This program requires a command line argument.");
else
{
int port =Integer.parseInt(args[0]);
while(j<6){
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.print("Message Received: ");
System.out.println(message);
Thread.sleep(10000);
System.out.println("Exiting..");
mySocket.close(); }
catch(Exception ex)
{ ex.printStackTrace();
}


 }
  }
    }
```

- **Start the two receivers first**

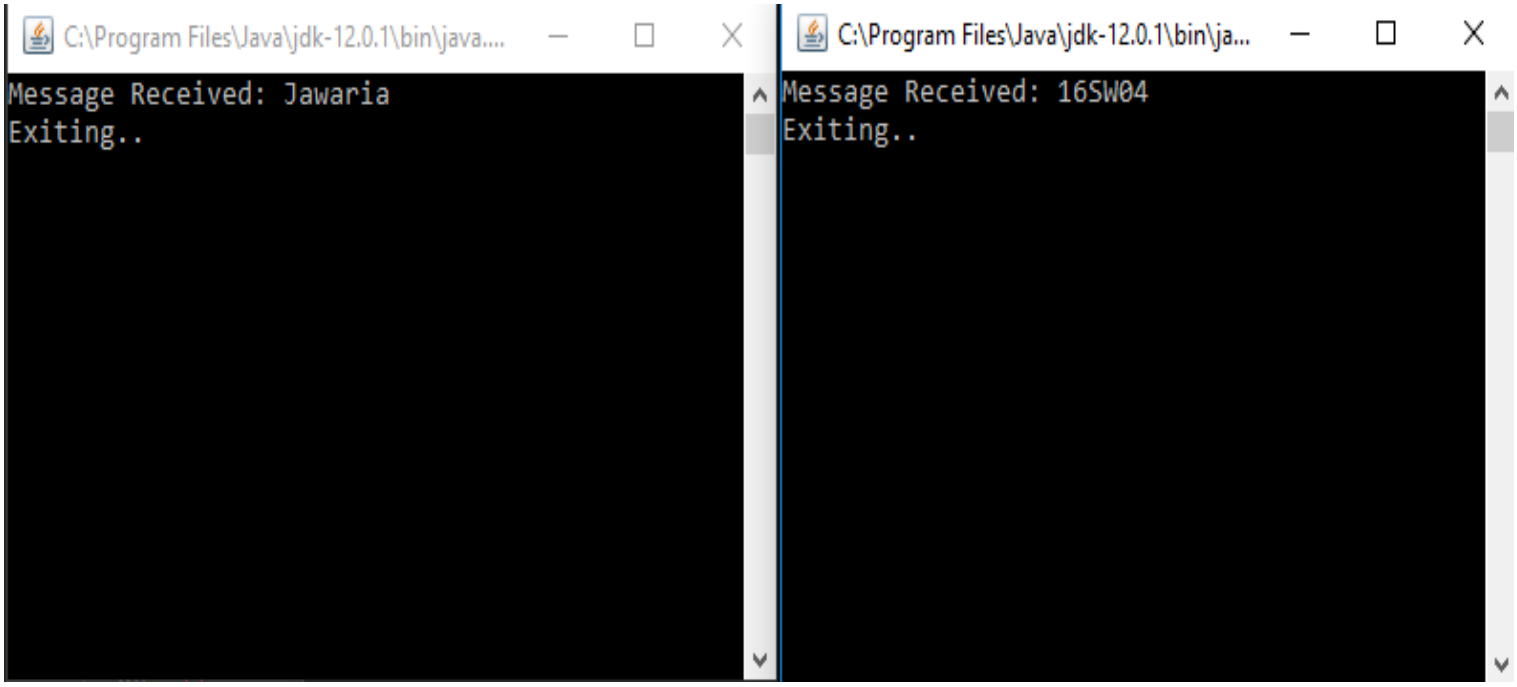
Command Prompt

```
E:\8th Semester\DC Labs\16SW04\Lab1>javac *.java

E:\8th Semester\DC Labs\16SW04\Lab1>start java Receiver 9090

E:\8th Semester\DC Labs\16SW04\Lab1>start java Receiver 8080

E:\8th Semester\DC Labs\16SW04\Lab1>
```

- **send message to two different receivers**



- **Yes, each receiver received the message.**



2. Modify the sample code so that the receiver loops five times to repeatedly receive then display the data received. Recompile. Then
   i. start the receiver
   ii. Execute the sender, sending a message “message1”, and
   iii. In another window, start another instance of the sender, sending a message “message2”.

Sender.java | Receiver.java | Client.java | Server.java

```java
import java.net.*;
import java.io.*;
public class Sender {
    public static void main(String[] args){
          // this application sends message using connectionless datagram socket
          // 16SW04
          // Jawaria Sattar
     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();
         }
     }  } }
```

Sender.java | Receiver.java | Client.java

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

public class Receiver{
public static void main(String[] args)
{
        // 16SW04
            // Jawaria Sattar
     int j=1;
if (args.length!=1)
System.out.println("This program requires a command line argument.");
else
{
int port =Integer.parseInt(args[0]);
while(j<6){
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.print("Message Received: ");
System.out.println(message);
Thread.sleep(10000);
System.out.println("Exiting..");
mySocket.close(); }
catch(Exception ex)
{ ex.printStackTrace();
}
j++;
}
 }
  }
   }
```

Command Prompt

```
E:\8th Semester\DC Labs\16SW04\Lab1>javac *.java

E:\8th Semester\DC Labs\16SW04\Lab1>start java Receiver 9090

E:\8th Semester\DC Labs\16SW04\Lab1>
```

Command Prompt

```
C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 Jawaria
C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 Sattar
C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 Dhakhan
C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 16SW04
C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 DC-Lab1
C:\Users\Haier\Desktop\16SW04>
```

C:\Program Files\Java\jdk-12.0.1\bin\java.exe

```
Message Received: Jawaria
Exiting..
Message Received: Sattar
Exiting..
Message Received: Dhakhan
Exiting..
Message Received: 16SW04
Exiting..
Message Received: DC-Lab1
```

Recompile:

i. **Start the receiver**

```
Command Prompt

E:\8th Semester\DC Labs\16SW04\Lab1>javac *.java

E:\8th Semester\DC Labs\16SW04\Lab1>start java Receiver 9090

E:\8th Semester\DC Labs\16SW04\Lab1>
```

ii. **Execute the sender, sending a message "message1", and**

iii. **In another window, start another instance of the sender, sending a message "message2".**

```
Command Prompt

C:\Users\Haier\Desktop\16SW04>javac *.java

C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 Jawaria

C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 Sattar

C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 16SW04

C:\Users\Haier\Desktop\16SW04>
```

```
Command Prompt

C:\Users\Haier\Desktop\16SW04>javac *.java

C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 DC-Lab1

C:\Users\Haier\Desktop\16SW04>java Sender 192.168.8.20 9090 Dhakhan

C:\Users\Haier\Desktop\16SW04>
```

C:\Program Files\Java\jdk-12.0.1\bin\java.exe

```
Message Received: DC-Lab1
Exiting..
Message Received: Jawaria
Exiting..
Message Received: Sattar
Exiting..
Message Received: Dhakhan
Exiting..
Message Received: 16SW04
```

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 | Sender.java | Receiver.java | Server.java

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

class Client
{
   public static void main(String args[]) throws Exception
   {
      // 16SW04
         // Jawaria Sattar
      BufferedReader inFromUser =
         new BufferedReader(new InputStreamReader(System.in));
      DatagramSocket clientSocket = new DatagramSocket();
      InetAddress IPAddress = InetAddress.getByName("localhost");
      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[10];
      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
      DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
      clientSocket.send(sendPacket);
      DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
      clientSocket.receive(receivePacket);
      String modifiedSentence = new String(receivePacket.getData());
      System.out.println("FROM SERVER:" + modifiedSentence+ "16SW04");
      clientSocket.close();
   }
}
```

Server.java | Client.java | Sender.java | Receiver.java

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

class Server
{
   public static void main(String args[]) throws Exception
      {
         // 16SW04
         // Jawaria Sattar
        DatagramSocket serverSocket = new DatagramSocket(9876);
           byte[] receiveData = new byte[1024];
           byte[] sendData = new byte[10];
           while(true)
              {
                 DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
                 serverSocket.receive(receivePacket);
                 String sentence = new String( receivePacket.getData());
                 System.out.println("RECEIVED: " + sentence);
                 InetAddress IPAddress = receivePacket.getAddress();
                 int port = receivePacket.getPort();
                 String capitalizedSentence = sentence.toUpperCase();
                 sendData = capitalizedSentence.getBytes();
                 DatagramPacket sendPacket =
                 new DatagramPacket(sendData, sendData.length, IPAddress, port);
                 serverSocket.send(sendPacket);
              }
      }
}
```

Command Prompt

```
E:\8th Semester\DC Labs\16SW04\Lab1>javac *.java

E:\8th Semester\DC Labs\16SW04\Lab1>start java Server.java

E:\8th Semester\DC Labs\16SW04\Lab1>java Client.java
Jawaria
FROM SERVER:JAWARIA   16SW04

E:\8th Semester\DC Labs\16SW04\Lab1>
```

C:\Program Files\Java\jdk-12.0.1\bin\java.exe

```
RECEIVED: Jawaria
```

Bonus Tasks

For Lab#01 (+1)

1. Implement two simple programs using Java datagram sockets, which broadcasts and multicast your roll number to all or selected network nodes respectively. Code guidance for these tasks can be obtained from the following link:
   https://www.baeldung.com/java-broadcast-multicast

i. 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.
- As per IPv4 Protocol, a broadcast address is a logical address, on which devices connected to the network are enabled to receive packets. In our example, we use a particular IP address, 255.255.255.255, which is the broadcast address of the local network.
- By definition, routers connecting a local network to other networks don't forward packets sent to this default broadcast address. Later we also show how we can iterate through all NetworkInterfaces, and send packets to their respective broadcast addresses.
- First, we demonstrate how to broadcast a message. To this extent, we need to call the setBroadcast() method on the socket to let it know that the packet is to be broadcasted:

BroadcastReceiver.java | BroadcastClient.java | Sender.java | Receiver.java | Sender1.java

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

public class BroadcastClient {
    private static DatagramSocket socket = null;

    public static void main(String[] args) throws IOException {
        broadcast("16SW04", 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, 4444);
        socket.send(packet);
        socket.close();
    }
```

- Following snippet shows how to iterate through all *NetworkInterfaces* to find their broadcast address:

BroadcastReceiver.java | BroadcastClient.java | Sender.java

```java

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;
}
}
```

Command Prompt

```
E:\8th Semester\DC Labs\16SW04\Lab1>javac *.java

E:\8th Semester\DC Labs\16SW04\Lab1>start java BroadcastReceiver 4444

E:\8th Semester\DC Labs\16SW04\Lab1>java BroadcastClient

E:\8th Semester\DC Labs\16SW04\Lab1>
```

C:\Program Files\Java\jdk-12.0.1\bin\java.exe

```
Message Received: 16SW04
```

ii. 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 solves this problem and sends packets to only those consumers who are interested. Multicasting is based on a group membership concept, where a multicast address represents each group.
- In IPv4, any address between 224.0.0.0 to 239.255.255.255 can be used as a multicast address. Only those nodes that subscribe to a group receive packets communicated to the group.
- In Java, *MulticastSocket* is used to receive packets sent to a multicast IP. The following example demonstrates the usage of *MulticastSocket*:

MulticastReceiver.java × Receiver1.java × BroadcastReceiver.java × BroadcastClient.java ×

```
java.net.*;
java.io.*;
class MulticastReceiver extends Thread {
cected static MulticastSocket socket = null;
cected static byte[] buf = new byte[256];

lic static void main(String[] args ) throws IOException {
 socket = new MulticastSocket(4445);
 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.print(message);
     String received = new String(packet.getData(), 0, packet.getLength());
     if ("end".equals(received)) {
         break;
     }
 }
 socket.leaveGroup(group);
 socket.close();

```

- After binding the *MulticastSocket* to a port, we call the *joinGroup()* method, with the multicast IP as an argument. This is necessary to be able to receive the packets published to this group. The *leaveGroup()* method can be used to leave the group.
- The following example shows how to publish to a multicast IP:

MulticastPublisher.java × Receiver1.java × BroadcastReceiver.java × BroadcastClient.java ×

```java

import java.net.*;
import java.io.*;
public class MulticastPublisher {
 public static void main(String[] args) throws IOException{
    multicast("16SW04");
}
    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, 4445);
        socket.send(packet);
        socket.close();
    }

}
```

Command Prompt

```
E:\8th Semester\DC Labs\16SW04\Lab1>javac *.java

E:\8th Semester\DC Labs\16SW04\Lab1>start java MulticastReceiver

E:\8th Semester\DC Labs\16SW04\Lab1>start java MulticastReceiver

E:\8th Semester\DC Labs\16SW04\Lab1>start java MulticastReceiver

E:\8th Semester\DC Labs\16SW04\Lab1>start java MulticastReceiver

E:\8th Semester\DC Labs\16SW04\Lab1>java MulticastPublisher

E:\8th Semester\DC Labs\16SW04\Lab1>
```

C:\Program Files\Java\jdk-12.0.1\bin\...

```
16SW04
```

16SW04

C:\Program Files\Java\jdk-12.0.1\bin\...

16SW04

C:\Program Files\Java\jdk-12.0.1\bin\java.exe

16SW04