

Assignment 1 A

Roll no: 43144

Name: Ruchika Pande

Title: Simple calculator using socket programming

Problem Statement:

To develop a simple calculator through implementing client-server communication programs based on JAVA Sockets.

Objectives:

- To study about client-server communication
- To study socket API for TCP and UDP

Theory:

Socket:

A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. A socket is a handle that a local program can pass to the networking API to connect to another machine. It can be defined as the terminal of a communication link through which two programs /processes/threads running on the network can communicate with each other. The TCP layer can easily identify the application location and access information through the port number assigned to the respective sockets. During an instance of communication, a client program creates a socket at its end and tries to connect it to the socket on the server. When the connection is made, the server creates a socket at its end and then server and client communication is established

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols –

- TCP – TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- UDP – UDP stands for User Datagram Protocol, a connectionless protocol that allows for packets of data to be transmitted between applications.

Socket Programming for TCP:

TCP is a Network Protocol which stands for Transfer Control Protocol, that allows well founded communication between applications. TCP is consistently used over the Internet Protocol, and that is why it is referred as TCP/IP. The communication mechanism between two systems, using TCP, can be established using Sockets and is known as Socket Programming. Hence, socket programming is a concept of Network Programming, that suggests to write programs that are executed across multiple systems, which are connected to each other using a network.

Mechanism for Socket Programming for TCP

A client creates a socket at it's end of transmission, and strives to connect the socket to the server. When a connection is established, the server creates a socket at it's end and client and server can now ready communicate through writing and reading methods. Following is the elaborated procedure of what happens when a TCP connection is established:

1. An object of *ServerSocket* is instantiated, and the desired port number is specified, on which connection is going to take place.
2. The *accept* method of *ServerSocket* is invoked, in order to hold the server in listening mode. This method won't resume until a client is connected to the server through the given port number.
3. Now, on the client side, an object of *Socket* is instantiated, and the desired port number and IP address is specified for the connection.
4. An attempt is made, for connecting the client to the server using the specified IP address and port number. If an attempt is successful, the client is provided with a *Socket* that is capable of communicating to the respective

server, with write and read methods. If unsuccessful, a desired exception is raised.

5. Since a client is connected to the server, *the accept* method on the server side resumes, providing a *Socket* that is capable of communicating to the connected client.
6. Once the communication is completed, terminate the sockets on both, the server and the client side.

Now, communication is held using input/output streams of Sockets. The *InputStream* of the client is coupled with the *OutputStream* of server and *OutputStream* of client is coupled with the *InputStream* of server. Since, TCP is a two-way network protocol, hence information can flow through both the streams at the time.

Socket Programming for UDP:

DatagramSockets are Java's mechanism for network communication via UDP instead of TCP. Java provides DatagramSocket to communicate over UDP instead of TCP. It is also built on top of IP. DatagramSockets can be used to both send and receive packets over the Internet.

One of the examples where UDP is preferred over TCP is the live coverage of TV channels. In this aspect, we want to transmit as many frames to live audiences as possible without worrying about the loss of one or two frames. TCP being a reliable protocol adds its own overhead while transmission.

Another example where UDP is preferred is online multiplayer gaming. In games like counter-strike or call of duty, it is not necessary to relay all the information but the most important ones. It should also be noted that most of the applications in real life use a careful blend of both UDP and TCP; transmitting the critical data over TCP and the rest of the data via UDP.

Mechanism for Socket Programming for UDP:

1. For sending a packet via UDP, we should know 4 things, the message to send, its length, ip address of destination, port at which destination is listening.
2. Once we know all these things, we can create the socket object for carrying the packets and packets which actually possess the data.
3. Invoke send()/receive() call for actually sending/receiving packets.

4. Extract the data from the received packet.

Code:

Client-Server communication using TCP:

Client Side Programming:

```
import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.InetAddress;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.Scanner;


public class client

{

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

    {

        InetAddress ip = InetAddress.getLocalHost();

        int port = 4444;

        Scanner sc = new Scanner(System.in);


        // Open the socket connection.
```

```
Socket s = new Socket(ip, port);

// the input and output stream

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

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

while (true)
{
    // Enter the equation in the form-
    // "operand1 operation operand2"

    System.out.print("Enter the equation in the form: ");

    System.out.println("operand operator operand");

    String inp = sc.nextLine();

    if (inp.equals("bye"))
        break;

    // send the equation to server

    dos.writeUTF(inp);

    // wait till request is processed and sent back to client
```

```
String ans = dis.readUTF();

System.out.println("Answer=" + ans);

}

}

}
```

Server Side Programming:

```
import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.StringTokenizer;

public class server

{

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

    {

        // Establishing the socket connection.

        ServerSocket ss = new ServerSocket(4444);
```

```
Socket s = ss.accept();

// Processing the request.

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

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

while (true)
{

    String input = dis.readUTF();

    if(input.equals("bye"))
        break;

    System.out.println("Equation received: " + input);

    int result;

    // StringTokenizer to break the equation into operand and
    // operator

    StringTokenizer st = new StringTokenizer(input);

    int oprnd1 = Integer.parseInt(st.nextToken());
```

```
String operation = st.nextToken();

int oprnd2 = Integer.parseInt(st.nextToken());

// perform the required operation.

if (operation.equals("+"))

{

    result = oprnd1 + oprnd2;

}

else if (operation.equals("-"))

{

    result = oprnd1 - oprnd2;

}

else if (operation.equals("*"))

{

    result = oprnd1 * oprnd2;

}

else

{

    result = oprnd1 / oprnd2;

}

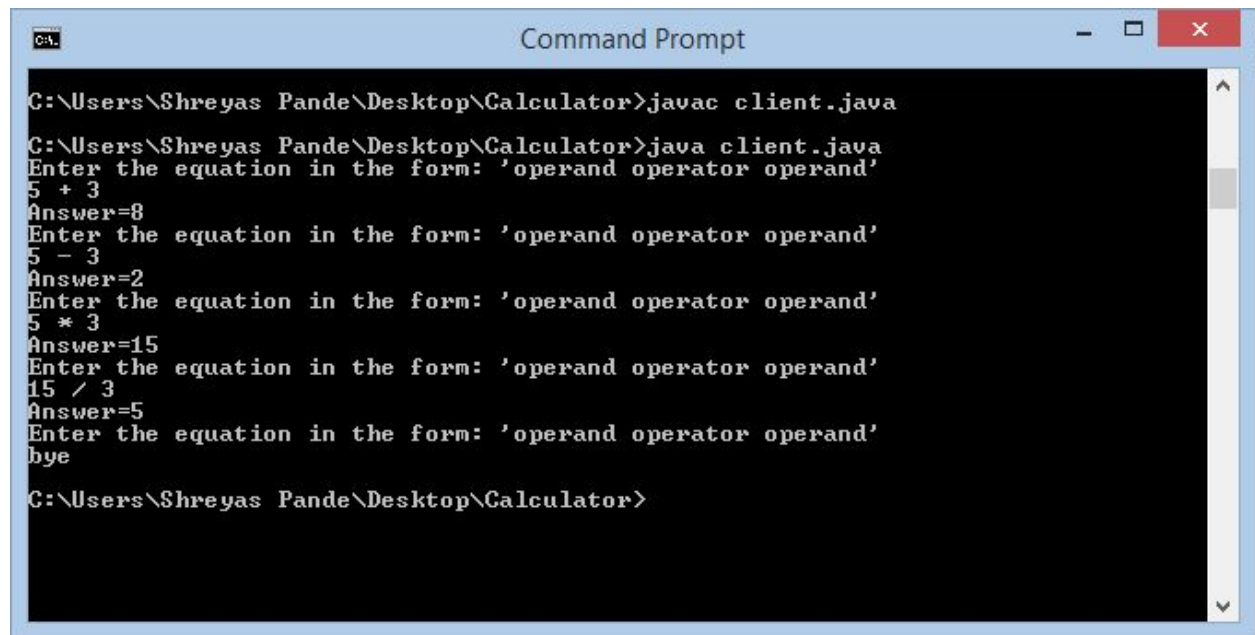
System.out.println("Sending the result...");
```



```
    // send the result back to the client.  
  
    dos.writeUTF(Integer.toString(result));  
  
    }  
  
}  
  
}
```

Output for TCP:

Client Side Terminal:



```
C:\Users\Shreyas Pande\Desktop\Calculator>javac client.java  
C:\Users\Shreyas Pande\Desktop\Calculator>java client.java  
Enter the equation in the form: 'operand operator operand'  
5 + 3  
Answer=8  
Enter the equation in the form: 'operand operator operand'  
5 - 3  
Answer=2  
Enter the equation in the form: 'operand operator operand'  
5 * 3  
Answer=15  
Enter the equation in the form: 'operand operator operand'  
15 / 3  
Answer=5  
Enter the equation in the form: 'operand operator operand'  
bye  
C:\Users\Shreyas Pande\Desktop\Calculator>
```

Server Side Terminal:

```
Command Prompt

C:\Users\Shreyas Pande\Desktop\Calculator>javac server.java
C:\Users\Shreyas Pande\Desktop\Calculator>java server.java
Equation received: 5 + 3
Sending the result...
Equation received: 5 - 3
Sending the result...
Equation received: 5 * 3
Sending the result...
Equation received: 15 / 3
Sending the result...
Exception in thread "main" java.net.SocketException: Connection reset
    at java.base/sun.nio.ch.NioSocketImpl.implRead(NioSocketImpl.java:324)
    at java.base/sun.nio.ch.NioSocketImpl.read(NioSocketImpl.java:351)
    at java.base/sun.nio.ch.NioSocketImpl$1.read(NioSocketImpl.java:802)
    at java.base/java.net.Socket$SocketInputStream.read(Socket.java:919)
    at java.base/java.net.Socket$SocketInputStream.read(Socket.java:914)
    at java.base/java.io.DataInputStream.readUnsignedShort(DataInputStream.java:342)
    at java.base/java.io.DataInputStream.readUTF(DataInputStream.java:594)
    at java.base/java.io.DataInputStream.readUTF(DataInputStream.java:569)
    at server.main(server.java:26)

C:\Users\Shreyas Pande\Desktop\Calculator>
```

Multithreaded client-server communication :

Multithreaded SocketServer :

```
import java.net.*;

import java.io.*;

public class MultithreadedSocketServer {

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

        try{

            ServerSocket server=new ServerSocket(8888);

            int counter=0;

            System.out.println("Server Started ....");

            while(true){

                counter++;

                Socket serverClient=server.accept(); //server accept the client
connection request
```

```

        System.out.println(">> " + "Client No:" + counter + " started!");

        ServerClientThread sct = new
ServerClientThread(serverClient,counter); //send the request to a
separate thread

        sct.start();

    }

    }catch(Exception e){

        System.out.println(e);

    }

}

}

```

ServerClientThread :

```

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.net.Socket;

import java.util.StringTokenizer;

class ServerClientThread extends Thread {

    Socket serverClient;

    int clientNo;

    int squire;

    ServerClientThread(Socket inSocket,int counter){

        serverClient = inSocket;

        clientNo=counter;
    }
}

```

```

    }

    public void run() {

        try {

            DataInputStream    inStream    =    new
DataInputStream(serverClient.getInputStream());

            DataOutputStream    outStream    =    new
DataOutputStream(serverClient.getOutputStream());

            String clientMessage="", serverMessage="";

            while(!clientMessage.equals("bye")) {

                clientMessage=inStream.readUTF();

                System.out.println("From Client-" +clientNo+ ": Equation received
is :"+clientMessage);

                int result;

                StringTokenizer st = new StringTokenizer( clientMessage);

                int oprnd1 = Integer.parseInt(st.nextToken());

                String operation = st.nextToken();

                int oprnd2 = Integer.parseInt(st.nextToken());

                // perform the required operation.

                if (operation.equals("+"))

                {

                    result = oprnd1 + oprnd2;

                }

```

```
else if (operation.equals("-"))
{
    result = oprnd1 - oprnd2;
}

else if (operation.equals("*"))
{
    result = oprnd1 * oprnd2;
}

else
{
    result = oprnd1 / oprnd2;
}

System.out.println("Sending the result...");

    serverMessage="From Server to Client-" + clientNo + " Result of "
+ clientMessage + " is " +result;

    outputStream.writeUTF(serverMessage);

    outputStream.flush();
}

inStream.close();

outStream.close();

serverClient.close();
```

```

    } catch (Exception ex) {

        System.out.println(ex);

    } finally {

        System.out.println("Client -" + clientNo + " exit!! ");

    }

}

}

```

TCPClient:

```

import java.net.*;

import java.util.Scanner;

import java.io.*;

public class TCPClient {

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

        try {

            Socket socket = new Socket("127.0.0.1", 8888);

            DataInputStream inStream = new DataInputStream(socket.getInputStream());

            DataOutputStream outStream = new
DataOutputStream(socket.getOutputStream());

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

            Scanner sc = new Scanner(System.in);

            String clientMessage = "", serverMessage = "";

            while (!clientMessage.equals("bye")) {

                System.out.println("Enter number :");

```

```
        System.out.print("Enter the equation in the form: ");

        System.out.println("'operand operator operand'");

        clientMessage = sc.nextLine();

        //clientMessage=br.readLine();

        outputStream.writeUTF(clientMessage);

        outputStream.flush();

        serverMessage=inStream.readUTF();

        System.out.println(serverMessage);

    }

    outputStream.close();

    outputStream.close();

    socket.close();

} catch (Exception e) {

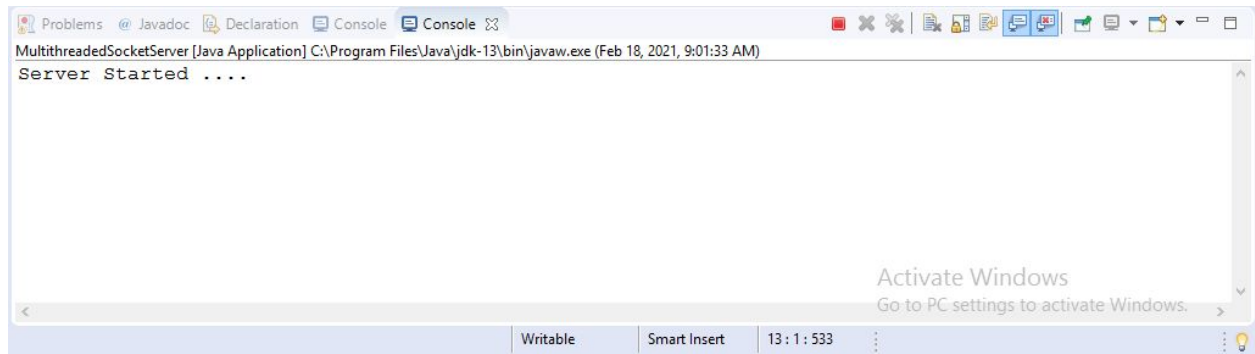
    System.out.println(e);

}

}
```

Output:

Server:

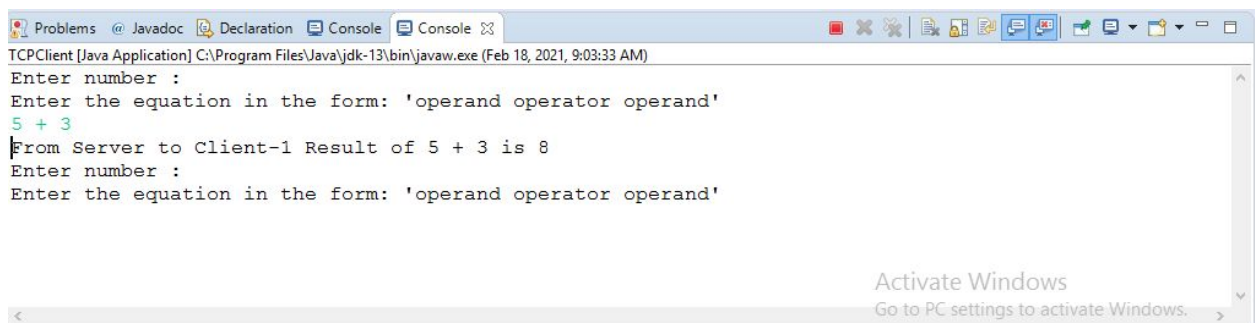


The screenshot shows the IDE's console window for the 'MultithreadedSocketServer' application. The output is 'Server Started'. The status bar at the bottom indicates 'Writable', 'Smart Insert', and the time '13:1:533'. An 'Activate Windows' watermark is visible in the bottom right corner.

```
Problems @ Javadoc Declaration Console Console
MultithreadedSocketServer [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (Feb 18, 2021, 9:01:33 AM)
Server Started ....

Activate Windows
Go to PC settings to activate Windows.
```

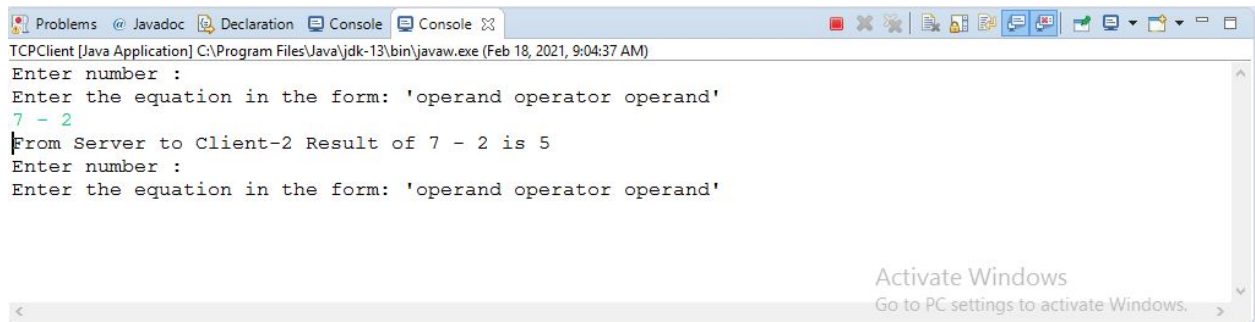
Clients:



The screenshot shows the IDE's console window for the 'TCPClient' application. The output shows the client entering '5 + 3' and receiving the result 'From Server to Client-1 Result of 5 + 3 is 8'. The status bar at the bottom indicates 'Writable', 'Smart Insert', and the time '13:1:533'. An 'Activate Windows' watermark is visible in the bottom right corner.

```
Problems @ Javadoc Declaration Console Console
TCPClient [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (Feb 18, 2021, 9:03:33 AM)
Enter number :
Enter the equation in the form: 'operand operator operand'
5 + 3
From Server to Client-1 Result of 5 + 3 is 8
Enter number :
Enter the equation in the form: 'operand operator operand'

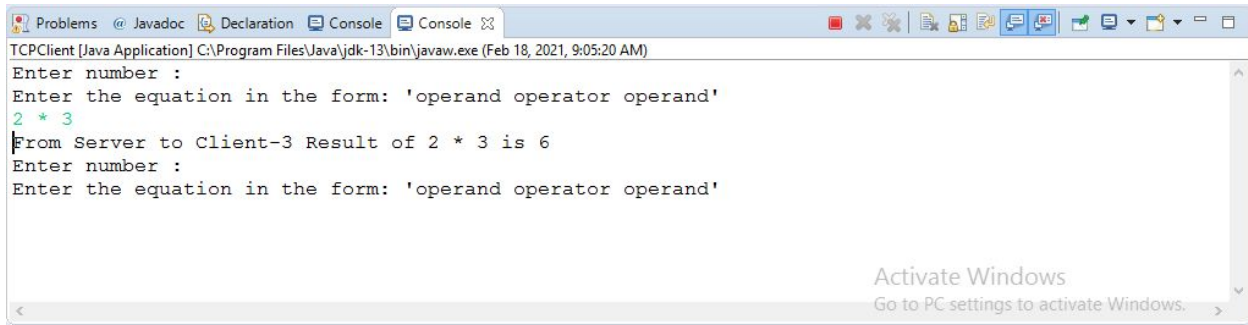
Activate Windows
Go to PC settings to activate Windows.
```



The screenshot shows the IDE's console window for the 'TCPClient' application. The output shows the client entering '7 - 2' and receiving the result 'From Server to Client-2 Result of 7 - 2 is 5'. The status bar at the bottom indicates 'Writable', 'Smart Insert', and the time '13:1:533'. An 'Activate Windows' watermark is visible in the bottom right corner.

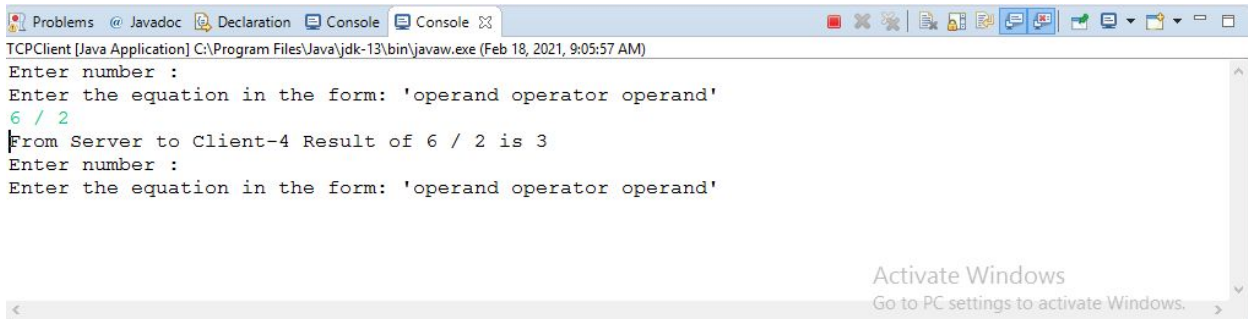
```
Problems @ Javadoc Declaration Console Console
TCPClient [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (Feb 18, 2021, 9:04:37 AM)
Enter number :
Enter the equation in the form: 'operand operator operand'
7 - 2
From Server to Client-2 Result of 7 - 2 is 5
Enter number :
Enter the equation in the form: 'operand operator operand'

Activate Windows
Go to PC settings to activate Windows.
```

The screenshot shows a Java IDE window with a console tab. The title bar indicates the application is 'TCPClient [Java Application]' running at 'C:\Program Files\Java\jdk-13\bin\javaw.exe' on 'Feb 18, 2021, 9:05:20 AM'. The console output shows a sequence of prompts and responses: 'Enter number :', 'Enter the equation in the form: 'operand operator operand'', the input '2 * 3', and the output 'From Server to Client-3 Result of 2 * 3 is 6'. This is followed by another prompt and input sequence. An 'Activate Windows' watermark is visible in the bottom right corner.

```
Enter number :
Enter the equation in the form: 'operand operator operand'
2 * 3
From Server to Client-3 Result of 2 * 3 is 6
Enter number :
Enter the equation in the form: 'operand operator operand'
```



This screenshot is similar to the one above, showing the same Java IDE console window. The title bar now shows the time as 'Feb 18, 2021, 9:05:57 AM'. The console output continues from the previous state, showing the input '6 / 2' and the output 'From Server to Client-4 Result of 6 / 2 is 3'. The 'Activate Windows' watermark is also present.

```
Enter number :
Enter the equation in the form: 'operand operator operand'
6 / 2
From Server to Client-4 Result of 6 / 2 is 3
Enter number :
Enter the equation in the form: 'operand operator operand'
```

Client-Server communication using UDP:

Client Side Programming:

```
import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.Scanner;

public class client

{

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

```
{

    Scanner sc = new Scanner(System.in);

    // Creating socket

    DatagramSocket ds = new DatagramSocket();

    InetAddress ip = InetAddress.getLocalHost();

    byte buf[] = null;

    while (true)
    {
        System.out.print("Enter the equation in the format:");

        System.out.println("operand1 operator operand2");

        String inp = sc.nextLine();

        buf = new byte[65535];

        // String to byte

        buf = inp.getBytes();

        // datagramPacket for sending the data.

        DatagramPacket DpSend =
```

```
        new DatagramPacket(buf, buf.length, ip, 7775);

        // send the data.
        ds.send(DpSend);

        if (inp.equals("bye"))
            break;

        buf = new byte[65535];

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

        ds.receive(DpReceive);

        System.out.println("Answer = " +
            new String(buf,0,buf.length));
    }
}
}
```

Server Side Programming:

```
import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.StringTokenizer;


public class server

{

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

    {

        DatagramSocket ds = new DatagramSocket(7775);

        byte[] buf = null;

        DatagramPacket DpReceive = null;

        DatagramPacket DpSend = null;

        while (true)

        {

            buf = new byte[65535];

            // DatagramPacket to receive the data.
```

```
DpReceive = new DatagramPacket(buf, buf.length);
```

```
// receive the data in byte buffer.
```

```
ds.receive(DpReceive);
```

```
String inp = new String(buf, 0, buf.length);
```

```
//To remove extra spaces.
```

```
inp=inp.trim();
```

```
System.out.println("Equation Received:- " + inp);
```

```
// Exit the server if the client sends "bye"
```

```
if (inp.equals("bye"))
```

```
{
```

```
    System.out.println("Client sent bye.....EXITING");
```

```
    break;
```

```
}
```

```
int result;
```

```
// Use StringTokenizer to break the
```

```
// equation into operand and operation
```

```
StringTokenizer st = new StringTokenizer(inp);
```

```
int oprnd1 = Integer.parseInt(st.nextToken());
```

```
String operation = st.nextToken();
```

```
int oprnd2 = Integer.parseInt(st.nextToken());
```

```
// perform the required operation.
```

```
if (operation.equals("+"))
```

```
    result = oprnd1 + oprnd2;
```

```
else if (operation.equals("-"))
```

```
    result = oprnd1 - oprnd2;
```

```
else if (operation.equals("*"))
```

```
    result = oprnd1 * oprnd2;
```

```
else
```

```
    result = oprnd1 / oprnd2;
```

```
System.out.println("Sending the result...");
```

```
String res = Integer.toString(result);
```

```

        buf = res.getBytes();

        // get the port of client.

        int port = DpReceive.getPort();

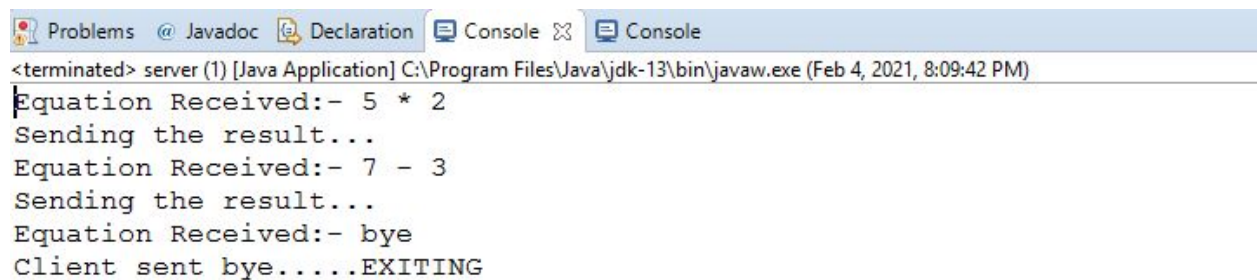
        DpSend = new DatagramPacket(buf, buf.length,
                                     InetAddress.getLocalHost(), port);

        ds.send(DpSend);
    }
}
}

```

Output for UDP:

Server-side terminal:



```

<terminated> server (1) [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (Feb 4, 2021, 8:09:42 PM)
Equation Received:- 5 * 2
Sending the result...
Equation Received:- 7 - 3
Sending the result...
Equation Received:- bye
Client sent bye.....EXITING

```

Client-side terminal:

```
client (1) [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (Feb 4, 2021, 8:09:32 PM)
Answer = 10
Enter the equation in the format: 'operand1 operator operand2 '
7 - 3
Answer = 4
Enter the equation in the format: 'operand1 operator operand2 '
7 - 3
Answer = 4
```

Conclusion:

In this assignment, we learned about client-server communication through different protocols and sockets. We also learned about Java support through the socket API for TCP and UDP.