

**Title:** To implement a RPC for calculating ADD(), SUB(), MUL(), DIV () using basics of Socket programming

1) 2.1 RPC Program

Server:

Code:

```
package lab_02;
```

```
import java.util.*;
```

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

```
import java.time.LocalDateTime;
```

```
import java.time.format.DateTimeFormatter;
```

```
public class RPCServer {
```

```
    DatagramSocket dataSocket;
```

```
    DatagramPacket dataPacket;
```

```
    String str, methodName, result;
```

```
    int val1, val2;
```

```
    RPCServer() {
```

```
        try {
```

```
            dataSocket = new DatagramSocket(1200);
```

```
            byte b[] = new byte[4096];
```

```
            while (true) {
```

```
dataPacket = new DatagramPacket(b, b.length);

dataSocket.receive(dataPacket);

str = new String(dataPacket.getData(), 0,
dataPacket.getLength());

if (str.equalsIgnoreCase("q")) {

    System.exit(1);

} else {

    StringTokenizer st = new StringTokenizer(str, " ");

    while (st.hasMoreTokens()) {

        methodName = st.nextToken();

        if (methodName.equalsIgnoreCase("date")) {

            DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");

            LocalDateTime now =
LocalDateTime.now();

            String date = "{" + dtf.format(now) +

            "},";

            result = "" + date;

            break;

        }

        val1 = Integer.parseInt(st.nextToken());

        if (st.hasMoreTokens()) {

            val2 =
Integer.parseInt(st.nextToken());

        }

    }

}
```

```
}  
  
System.out.println(str);  
  
if (methodName.equalsIgnoreCase("add")) {  
    result = "" + add(val1, val2);  
} else if (methodName.equalsIgnoreCase("sub")) {  
    result = "" + sub(val1, val2);  
} else if (methodName.equalsIgnoreCase("mul")) {  
    result = "" + mul(val1, val2);  
} else if (methodName.equalsIgnoreCase("div")) {  
    result = "" + div(val1, val2);  
} else if (methodName.equalsIgnoreCase("mod")) {  
    result = "" + mod(val1, val2);  
} else if (methodName.equalsIgnoreCase("pow")) {  
    result = "" + pow(val1, val2);  
} else if (methodName.equalsIgnoreCase("sqrt")) {  
    result = "" + sqrt(val1);  
} else if (methodName.equalsIgnoreCase("log")) {  
    result = "" + log(val1);  
} else if (methodName.equalsIgnoreCase("abs")) {  
    result = "" + abs(val1);  
} else if (methodName.equalsIgnoreCase("fact")) {  
    result = "" + fact(val1);  
} else if (methodName.equalsIgnoreCase("cbrt")) {  
    result = "" + cbrt(val1);
```

```
        } else if (methodName.equalsIgnoreCase("sin")) {  
            result = "" + sin(val1);  
        } else if (methodName.equalsIgnoreCase("cos")) {  
            result = "" + cos(val1);  
        } else if (methodName.equalsIgnoreCase("tan")) {  
            result = "" + tan(val1);  
        } else if (methodName.equalsIgnoreCase("expo")) {  
            result = "" + expo(val1);  
        } else if (methodName.equalsIgnoreCase("min")) {  
            result = "" + min(val1, val2);  
        } else if (methodName.equalsIgnoreCase("max")) {  
            result = "" + max(val1, val2);  
        }  
  
        byte b1[] = result.getBytes();  
  
        DatagramSocket ds1 = new DatagramSocket();  
  
        DatagramPacket dp1 = new DatagramPacket(b1, b1.length,  
        InetAddress.getLocalHost(), 1300);  
  
        System.out.println("result : " + result + "\n");  
  
        ds1.send(dp1);  
  
        ds1.close();  
    }  
  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
public int add(int val1, int val2) {  
    return val1 + val2;  
}
```

```
public int abs(int val1) {  
    return Math.abs(val1);  
}
```

```
public int sub(int val1, int val2) {  
    return val1 - val2;  
}
```

```
public int mul(int val1, int val2) {  
    return val1 * val2;  
}
```

```
public int div(int val1, int val2) {  
    return val1 / val2;  
}
```

```
public int mod(int val1, int val2) {  
    return val1 % val2;  
}
```

```
public int pow(int val1, int val2) {  
    return (int) Math.pow(val1, val2);  
}
```

```
public double sqrt(int val1) {  
    return Math.sqrt(val1);  
}
```

```
public double cbrt(int val1) {  
    return Math.cbrt(val1);  
}
```

```
public double log(int val1) {  
    return Math.log(val1);  
}
```

```
public double sin(int val1) {  
    return Math.sin(Math.toRadians(val1));  
}
```

```
public double cos(int val1) {  
    return Math.cos(Math.toRadians(val1));  
}
```

```
public double tan(int val1) {  
    return Math.tan(Math.toRadians(val1));  
}  
  
public double expo(int val1) {  
    return Math.exp(val1);  
}  
  
public int min(int val1, int val2) {  
    return Math.min(val1, val2);  
}  
  
public int max(int val1, int val2) {  
    return Math.max(val1, val2);  
}  
  
public int fact(int val1) {  
    return (val1 == 1 || val1 == 0) ? 1 : val1 * fact(val1 - 1);  
}  
  
public static void main(String[] args) {  
    new RPCServer();  
}
```

```
}
```

```
package lab_02;
```

```
import java.util.*;
```

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

```
import java.time.LocalDateTime;
```

```
import java.time.format.DateTimeFormatter;
```

```
public class RPCServer {
```

```
    DatagramSocket dataSocket;
```

```
    DatagramPacket dataPacket;
```

```
    String str, methodName, result;
```

```
    int val1, val2;
```

```
    RPCServer() {
```

```
        try {
```

```
            dataSocket = new DatagramSocket(1200);
```

```
            byte b[] = new byte[4096];
```

```
            while (true) {
```

```
                dataPacket = new DatagramPacket(b, b.length);
```

```
                dataSocket.receive(dataPacket);
```

```
                str = new String(dataPacket.getData(), 0,  
dataPacket.getLength());
```

```
                if (str.equalsIgnoreCase("q")) {
```

Hammad Ansari

2018450002



```
        System.exit(1);
    } else {
        StringTokenizer st = new StringTokenizer(str, " ");
        while (st.hasMoreTokens()) {
            methodName = st.nextToken();
            if (methodName.equalsIgnoreCase("date")) {
                DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");

                LocalDateTime now =
LocalDateTime.now();

                String date = "{" + dtf.format(now) +

"}";

                result = "" + date;
                break;
            }
            val1 = Integer.parseInt(st.nextToken());
            if (st.hasMoreTokens()) {
                val2 =
Integer.parseInt(st.nextToken());

            }
        }
    }

    System.out.println(str);

    if (methodName.equalsIgnoreCase("add")) {
        result = "" + add(val1, val2);
    } else if (methodName.equalsIgnoreCase("sub")) {
```

```
        result = "" + sub(val1, val2);
    } else if (methodName.equalsIgnoreCase("mul")) {
        result = "" + mul(val1, val2);
    } else if (methodName.equalsIgnoreCase("div")) {
        result = "" + div(val1, val2);
    } else if (methodName.equalsIgnoreCase("mod")) {
        result = "" + mod(val1, val2);
    } else if (methodName.equalsIgnoreCase("pow")) {
        result = "" + pow(val1, val2);
    } else if (methodName.equalsIgnoreCase("sqrt")) {
        result = "" + sqrt(val1);
    } else if (methodName.equalsIgnoreCase("log")) {
        result = "" + log(val1);
    } else if (methodName.equalsIgnoreCase("abs")) {
        result = "" + abs(val1);
    } else if (methodName.equalsIgnoreCase("fact")) {
        result = "" + fact(val1);
    } else if (methodName.equalsIgnoreCase("cbrt")) {
        result = "" + cbrt(val1);
    } else if (methodName.equalsIgnoreCase("sin")) {
        result = "" + sin(val1);
    } else if (methodName.equalsIgnoreCase("cos")) {
        result = "" + cos(val1);
    } else if (methodName.equalsIgnoreCase("tan")) {
```

```
        result = "" + tan(val1);
    } else if (methodName.equalsIgnoreCase("expo")) {
        result = "" + expo(val1);
    } else if (methodName.equalsIgnoreCase("min")) {
        result = "" + min(val1, val2);
    } else if (methodName.equalsIgnoreCase("max")) {
        result = "" + max(val1, val2);
    }

    byte b1[] = result.getBytes();

    DatagramSocket ds1 = new DatagramSocket();

    DatagramPacket dp1 = new DatagramPacket(b1, b1.length,
    InetAddress.getLocalHost(), 1300);

    System.out.println("result : " + result + "\n");

    ds1.send(dp1);

    ds1.close();

    }

    } catch (Exception e) {

        e.printStackTrace();

    }

}

public int add(int val1, int val2) {

    return val1 + val2;

}
```

```
public int abs(int val1) {  
    return Math.abs(val1);  
}
```

```
public int sub(int val1, int val2) {  
    return val1 - val2;  
}
```

```
public int mul(int val1, int val2) {  
    return val1 * val2;  
}
```

```
public int div(int val1, int val2) {  
    return val1 / val2;  
}
```

```
public int mod(int val1, int val2) {  
    return val1 % val2;  
}
```

```
public int pow(int val1, int val2) {  
    return (int) Math.pow(val1, val2);  
}
```

```
public double sqrt(int val1) {  
    return Math.sqrt(val1);  
}
```

```
public double cbrt(int val1) {  
    return Math.cbrt(val1);  
}
```

```
public double log(int val1) {  
    return Math.log(val1);  
}
```

```
public double sin(int val1) {  
    return Math.sin(Math.toRadians(val1));  
}
```

```
public double cos(int val1) {  
    return Math.cos(Math.toRadians(val1));  
}
```

```
public double tan(int val1) {  
    return Math.tan(Math.toRadians(val1));  
}
```

```
    public double expo(int val1) {  
        return Math.exp(val1);  
    }  
  
    public int min(int val1, int val2) {  
        return Math.min(val1, val2);  
    }  
  
    public int max(int val1, int val2) {  
        return Math.max(val1, val2);  
    }  
  
    public int fact(int val1) {  
        return (val1 == 1 || val1 == 0) ? 1 : val1 * fact(val1 - 1);  
    }  
  
    public static void main(String[] args) {  
        new RPCServer();  
    }  
}  
  
package lab_02;  
  
import java.util.*;
```

```
import java.net.*;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;


public class RPCServer {

    DatagramSocket dataSocket;

    DatagramPacket dataPacket;

    String str, methodName, result;

    int val1, val2;


    RPCServer() {

        try {

            dataSocket = new DatagramSocket(1200);

            byte b[] = new byte[4096];

            while (true) {

                dataPacket = new DatagramPacket(b, b.length);

                dataSocket.receive(dataPacket);

                str = new String(dataPacket.getData(), 0,
dataPacket.getLength());

                if (str.equalsIgnoreCase("q")) {

                    System.exit(1);

                } else {

                    StringTokenizer st = new StringTokenizer(str, " ");

                    while (st.hasMoreTokens()) {

                        methodName = st.nextToken();
```

```
        if (methodName.equalsIgnoreCase("date")) {

            DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");

            LocalDateTime now =
LocalDateTime.now();

            String date = "{" + dtf.format(now) +

"}";

            result = "" + date;

            break;

        }

        val1 = Integer.parseInt(st.nextToken());

        if (st.hasMoreTokens()) {

            val2 =

Integer.parseInt(st.nextToken());

        }

    }

}

System.out.println(str);

if (methodName.equalsIgnoreCase("add")) {

    result = "" + add(val1, val2);

} else if (methodName.equalsIgnoreCase("sub")) {

    result = "" + sub(val1, val2);

} else if (methodName.equalsIgnoreCase("mul")) {

    result = "" + mul(val1, val2);

} else if (methodName.equalsIgnoreCase("div")) {

    result = "" + div(val1, val2);

}
```



```
} else if (methodName.equalsIgnoreCase("mod")) {  
    result = "" + mod(val1, val2);  
}  
} else if (methodName.equalsIgnoreCase("pow")) {  
    result = "" + pow(val1, val2);  
}  
} else if (methodName.equalsIgnoreCase("sqrt")) {  
    result = "" + sqrt(val1);  
}  
} else if (methodName.equalsIgnoreCase("log")) {  
    result = "" + log(val1);  
}  
} else if (methodName.equalsIgnoreCase("abs")) {  
    result = "" + abs(val1);  
}  
} else if (methodName.equalsIgnoreCase("fact")) {  
    result = "" + fact(val1);  
}  
} else if (methodName.equalsIgnoreCase("cbrt")) {  
    result = "" + cbrt(val1);  
}  
} else if (methodName.equalsIgnoreCase("sin")) {  
    result = "" + sin(val1);  
}  
} else if (methodName.equalsIgnoreCase("cos")) {  
    result = "" + cos(val1);  
}  
} else if (methodName.equalsIgnoreCase("tan")) {  
    result = "" + tan(val1);  
}  
} else if (methodName.equalsIgnoreCase("expo")) {  
    result = "" + expo(val1);  
}  
} else if (methodName.equalsIgnoreCase("min")) {  
    result = "" + min(val1, val2);  
}
```

```
        } else if (methodName.equalsIgnoreCase("max")) {  
            result = "" + max(val1, val2);  
        }  
        byte b1[] = result.getBytes();  
        DatagramSocket ds1 = new DatagramSocket();  
        DatagramPacket dp1 = new DatagramPacket(b1, b1.length,  
        InetAddress.getLocalHost(), 1300);  
        System.out.println("result : " + result + "\n");  
        ds1.send(dp1);  
        ds1.close();  
    }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
  
public int add(int val1, int val2) {  
    return val1 + val2;  
}  
  
public int abs(int val1) {  
    return Math.abs(val1);  
}  
  
public int sub(int val1, int val2) {
```

```
        return val1 - val2;
    }
}
```

```
public int mul(int val1, int val2) {
    return val1 * val2;
}
```

```
public int div(int val1, int val2) {
    return val1 / val2;
}
```

```
public int mod(int val1, int val2) {
    return val1 % val2;
}
```

```
public int pow(int val1, int val2) {
    return (int) Math.pow(val1, val2);
}
```

```
public double sqrt(int val1) {
    return Math.sqrt(val1);
}
```

```
public double cbrt(int val1) {
```

```
        return Math.cbrt(val1);  
    }  
  
}
```

```
public double log(int val1) {  
    return Math.log(val1);  
}  
  
}
```

```
public double sin(int val1) {  
    return Math.sin(Math.toRadians(val1));  
}  
  
}
```

```
public double cos(int val1) {  
    return Math.cos(Math.toRadians(val1));  
}  
  
}
```

```
public double tan(int val1) {  
    return Math.tan(Math.toRadians(val1));  
}  
  
}
```

```
public double expo(int val1) {  
    return Math.exp(val1);  
}  
  
}
```

```
public int min(int val1, int val2) {
```

```
        return Math.min(val1, val2);
    }

    public int max(int val1, int val2) {
        return Math.max(val1, val2);
    }

    public int fact(int val1) {
        return (val1 == 1 || val1 == 0) ? 1 : val1 * fact(val1 - 1);
    }

    public static void main(String[] args) {
        new RPCServer();
    }
}
```

Screenshot:

```

RPCServer [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (25-Aug-2020, 6:00:46 pm)
date
result : 2020/08/25 18:00:53

date
result : 2020/08/25 18:01:22

date
result : 2020/08/25 18:02:33

add 3 4
result : 7

mu 3 4
result : 7

sqrt 4
result : 2.0

```

Client:

Code:

```
package lab_02;
```

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

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

```
class RPCCClient {
```

```
    RPCCClient() {
```

```
        try (DatagramSocket dataSocket = new DatagramSocket(); DatagramSocket
dataSocket1 = new DatagramSocket(1300);) {
```

```
            System.out.println("\nRPC Client\n");
```

```
            System.out.println("Methods:\n"
```

```
                + "1. Add\n2. Sub\n3. Mul\n4. Div\n5. Pow\n6.
```

```
Mod\n7. Sqrt\n8. Log\n9. Abs\n10. Fact\n11. Cube Root(cbrt)\n12. Sin\n13. Cos\n14.
```

```
Tan\n15. Expo\n16. Min\n17. Max\n");
```

```
            System.out.println("Enter method name and parameter like pow 9
3\n");
```

```
            while (true) {
```

Hammad Ansari

2018450002

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

        String str = br.readLine();

        byte b[] = str.getBytes();

        DatagramPacket dataPacket = new DatagramPacket(b,
b.length, InetAddress.getLocalHost(), 1200);

        dataSocket.send(dataPacket);

        dataPacket = new DatagramPacket(b, b.length);

        dataSocket1.receive(dataPacket);

        String s = new String(dataPacket.getData(), 0,
dataPacket.getLength());

        System.out.println("\nResult = " + s + "\n");

    }

    } catch (Exception e) {

        e.printStackTrace();

    }

}

public static void main(String[] args) {

    new RPCClient();

}

}
```

Screenshot:

```
RPCClient [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (25-Aug-2020, 6:05:38 pm)

RPC Client

Methods:
1. Add
2. Sub
3. Mul
4. Div
5. Pow
6. Mod
7. Sqrt
8. Log
9. Abs
10. Fact
11. Cube Root(cbrt)
12. Sin
13. Cos
14. Tan
15. Expo
16. Min
17. Max

Enter method name and parameter like pow 9 3

add 3 4
Result = 7

mul 3 4
Result = 7

sqrt 4
Result = 2.0
```

## 2) 2.2 RMI Calculator:

Server Interface:

```
public interface Calculator extends java.rmi.Remote {

    public long add(long a, long b) throws java.rmi.RemoteException;

    public long sub(long a, long b) throws java.rmi.RemoteException;

    public long mul(long a, long b) throws java.rmi.RemoteException;

    public long div(long a, long b) throws java.rmi.RemoteException;

    public double squareRoot(long a) throws java.rmi.RemoteException;

}
```

Hammad Ansari

2018450002



ServerImpl:

```
public class CalculatorImp extends java.rmi.server.UnicastRemoteObject implements  
Calculator {
```

```
    /**
```

```
     *
```

```
    */
```

```
    private static final long serialVersionUID = 1L;
```

```
    // Implementations must have an
```

```
    // explicit constructor
```

```
    // in order to declare the
```

```
    // RemoteException exception
```

```
    public CalculatorImp() throws java.rmi.RemoteException {
```

```
        super();
```

```
    }
```

```
    public long add(long a, long b) throws java.rmi.RemoteException {
```

```
        return a + b;
```

```
    }
```

```
    public long sub(long a, long b) throws java.rmi.RemoteException {
```

```
        return a - b;
```

```
    }
```

```
public long mul(long a, long b) throws java.rmi.RemoteException {  
    return a * b;  
}  
  
public long div(long a, long b) throws java.rmi.RemoteException {  
    return a / b;  
}  
  
public double squareRoot(long a) throws java.rmi.RemoteException {  
    return Math.sqrt(a);  
}  
}
```

Server:

```
import java.rmi.Naming;
```

```
public class CalculatorServer {  
  
    public CalculatorServer() {  
        try {  
            Calculator c = new CalculatorImp();  
            Naming.rebind("rmi://localhost:1099/CalculatorService", c);  
        } catch (Exception e) {  
            System.out.println("Trouble: " + e);  
        }  
    }  
}
```

```
    }  
}  
  
    public static void main(String args[]) {  
        new CalculatorServer();  
    }  
}
```

Client:

```
import java.rmi.Naming;  
import java.rmi.RemoteException;  
import java.net.MalformedURLException;  
import java.rmi.NotBoundException;  
  
public class CalculatorClient {  
  
    public static void main(String[] args) {  
        try {  
            Calculator c = (Calculator) Naming.lookup("rmi://localhost/CalculatorService");  
            long d1 = Long.valueOf(args[0]);  
            long d2 = Long.valueOf(args[1]);  
            System.out.println(c.sub(d1, d2));  
            System.out.println(c.add(d1, d2));  
            System.out.println(c.mul(d1, d2));  
        }  
    }  
}
```

```
        System.out.println(c.div(d1, d2));

        System.out.println(c.squareRoot(d1));
    } catch (MalformedURLException me) {

        System.out.println("MalformedURLException");

        System.out.println(me);
    } catch (RemoteException re) {

        System.out.println("RemoteException");

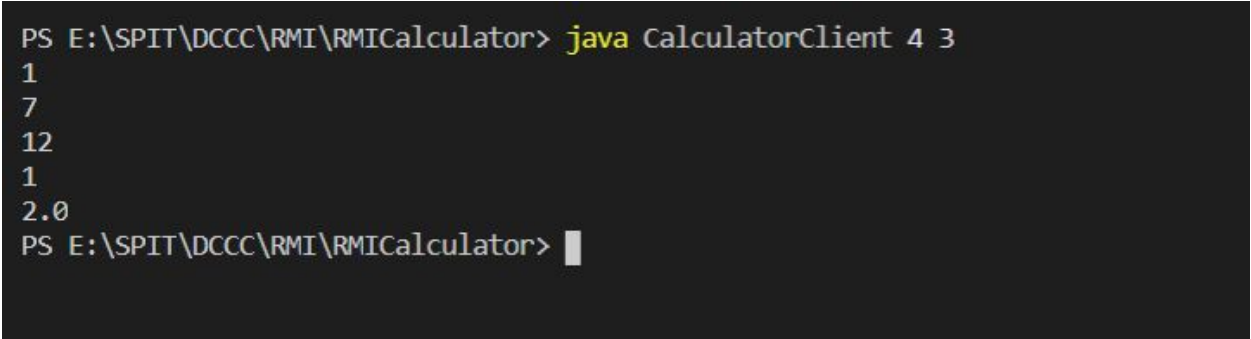
        System.out.println(re);
    } catch (NotBoundException nbe) {

        System.out.println("NotBoundException");

        System.out.println(nbe);
    } catch (java.lang.ArithmeticException ae) {

        System.out.println("java.lang.ArithmeticException");

        System.out.println(ae);
    }
}
}
```



```
PS E:\SPIT\DCCC\RFI\RFICalculator> java CalculatorClient 4 3
1
7
12
1
2.0
PS E:\SPIT\DCCC\RFI\RFICalculator> █
```

- 3) 2.3 Equation : The client should provide an equation to the server through an interface. The server will solve the expression given by the client.  $(a-b)^2 = a^2 - 2ab + b^2$

Equation:

```
import java.rmi.RemoteException;
```

```
import java.rmi.Remote;
```

```
public interface Equation extends Remote {
```

```
    double getEquationResult(double a, double b) throws RemoteException;
```

```
}
```

EquationImpl:

```
import java.rmi.*;
```

```
import java.rmi.server.*;
```

```
public class EquationImpl extends UnicastRemoteObject implements Equation {
```

```
    private static final long serialVersionUID = 1L;
```

```
    public EquationImpl() throws RemoteException {
```

```
    }
```

```
    public double getEquationResult(double a, double b) throws RemoteException {
```

```
        return ((a*a)+(b*b)-(2*a*b));
```

```
    }
```

```
}
```

EquationClient:

Hammad Ansari

2018450002

```
import java.rmi.*;

public class EquationClient {

    public static void main(String args[]) {

        try {

            String URL = "rmi://localhost/EquationServer";

            Equation equationSolver = (Equation) Naming.lookup(URL);

            System.out.println("a = " + args[0]);

            System.out.println("b = " + args[1]);

            double a = Double.valueOf(args[0]).doubleValue();

            double b = Double.valueOf(args[1]).doubleValue();

            System.out.println("The result of the equation is: " + equationSolver.getEquationResult(a,
b));

        } catch (Exception e) {

            System.out.println(e.getMessage());

        }

    }

}
```

EquationServer:

```
import java.rmi.Naming;

public class EquationServer {

    public static void main(String args[]) {

        try {

            EquationImpl eq = new EquationImpl();
```

Hammad Ansari

2018450002

```

        Naming.rebind("EquationServer", eq);

    } catch (Exception e) {

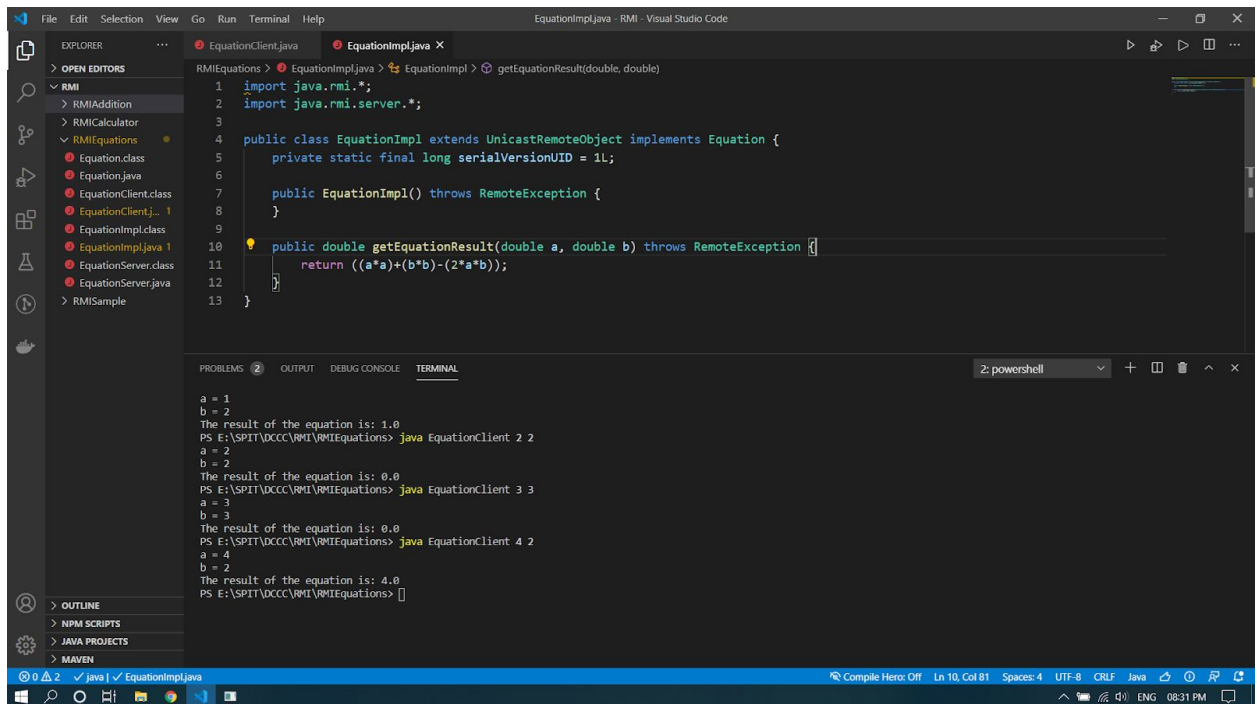
        System.out.println(e);

    }

}

}

```



#### 4) 2.4 Nishita's Program with RPC

Server.java

```
package Lab_5_RPC;
```

```
import java.io.DataInputStream;
```

```
import java.io.DataOutputStream;
```

Hammad Ansari

2018450002

```
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

    {


        // Step 1: Establish the socket connection.

        ServerSocket ss = new ServerSocket(4444);

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

        Socket s = ss.accept();


        // Step 2: Processing the request.

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

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


        while (true)

        {

            // wait for input

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



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

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

        int result;

        // Use StringTokenizer to break the equation into operand and
        // operation

        StringTokenizer st = new StringTokenizer(input);

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

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

        // perform the required operation.

        result = (int) (Math.pow(oprnd1, 2)+Math.pow(oprnd2, 2)-2*oprnd1*oprnd2);

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

        // send the result back to the client.

        dos.writeUTF(Integer.toString(result));

    }

}

}
```

Client.java

```
package Lab_5_RPC;
```

```
import java.io.DataInputStream;
```

```
import java.io.DataOutputStream;
```

```
import java.io.IOException;
```

```
import java.net.InetAddress;
```

```
import java.net.Socket;
```

```
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);
```

```
        // Step 1: Open the socket connection.
```

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

```
        // Step 2: Communication-get the input and output stream
```

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

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

Hammad Ansari

2018450002

```
while (true)
{
    // Enter the equation in the form-
    // "operand1 operation operand2"
    System.out.print("Enter the values of a and b: ");
    System.out.println("'value of a' 'value of b' ");

    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("(a-b)^2 = a^2 + b^2 - 2ab = " + ans);
}
}
}
```

OUTPUT:

Hammad Ansari

2018450002

```
<terminated> Server (4) [Java Application] C:\Program Files\Java\jre1.8.0_261\bin\javaw.exe (07-5
Server ready
Equation received:-3 9
Sending the result...
Equation received:--1 -5
Sending the result...
Equation received:--6 9
Sending the result...
Exception in thread "main" java.net.SocketException: Connection reset
    at java.net.SocketInputStream.read(Unknown Source)
    at java.net.SocketInputStream.read(Unknown Source)
    at java.net.SocketInputStream.read(Unknown Source)
    at java.io.DataInputStream.readUnsignedShort(Unknown Source)
    at ClientSocketHandler.handleClientRequest(SocketHandler.java:101)
    at ClientSocketHandler.run(SocketHandler.java:61)
    at java.lang.Thread.run(Thread.java:748)

<terminated> Client (5) [Java Application] C:\Program Files\Java\jre1.8.0_261\b
Enter the values of a and b: 'value of a' 'value of b'
3 9
(a-b)^2 = a^2 + b^2 - 2ab = 36
Enter the values of a and b: 'value of a' 'value of b'
-1 -5
(a-b)^2 = a^2 + b^2 - 2ab = 16
Enter the values of a and b: 'value of a' 'value of b'
-6 9
(a-b)^2 = a^2 + b^2 - 2ab = 225
Enter the values of a and b: 'value of a' 'value of b'
bye
```