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

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")) {
```

```
                    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);
```

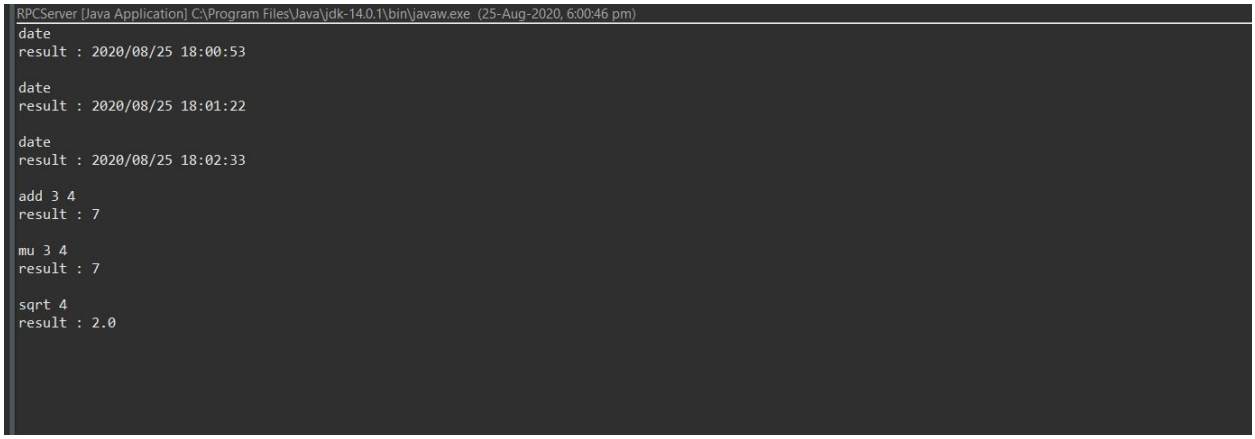
```
                                } else if (methodName.equalsIgnoreCase("max")) {
                                        result = "" + max(val1, val2);
                                }
                                byte b1[] = result.getBytes();
                                DatagramSocket ds1 = 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:



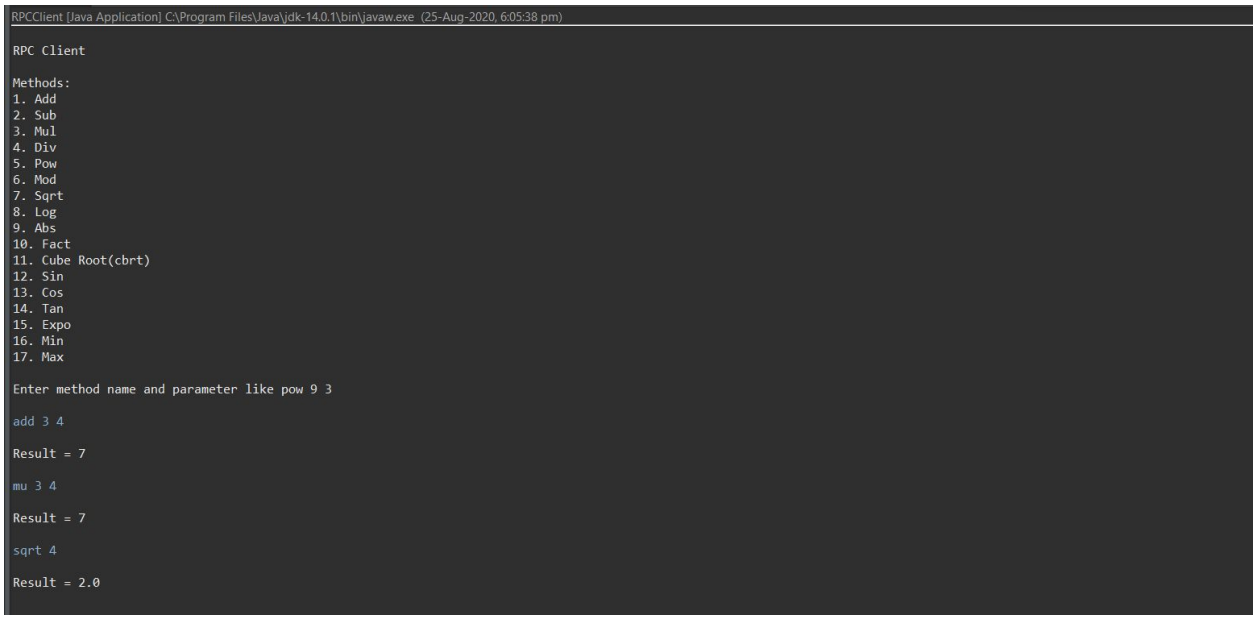
Client:

Code:

```
package lab_02;


import java.io.*;

import java.net.*;


class RPCClient {

	RPCClient() {

		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) {
```

```
                                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:



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;
}
```

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\RMI\RMICalculator> java CalculatorClient 4 3
1
7
12
1
2.0
PS E:\SPIT\DCCC\RMI\RMICalculator>
```

3) Equation : 2.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:

```
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();
```

```
        Naming.rebind("EquationServer", eq);
      } catch (Exception e) {
        System.out.println(e);
      }
    }
}
```