

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

1) 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);

    }

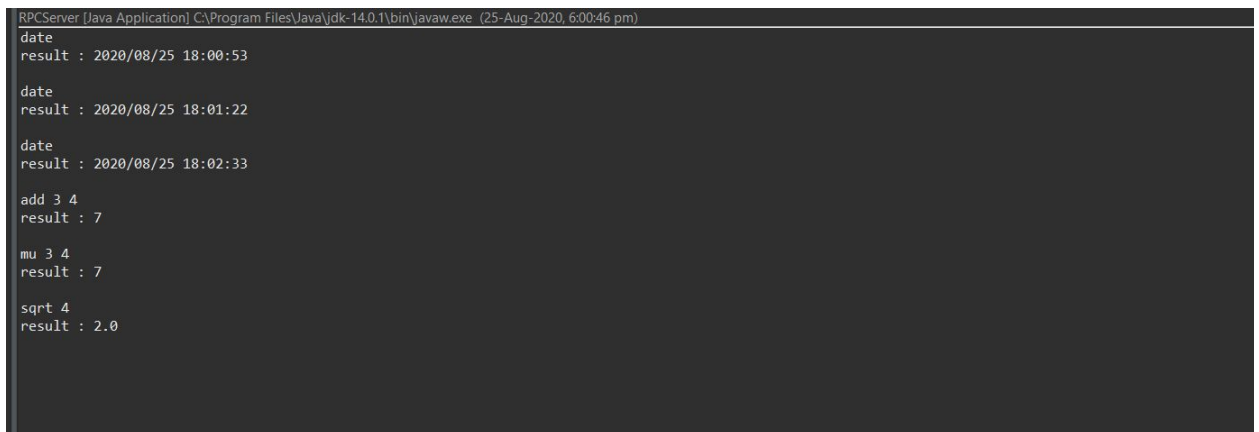
    public static void main(String[] args) {

        new RPCServer();

    }

}
```

Screenshot:



```
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
```

2) 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:

```
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
```