Experiment No. 2

Aim: Write a program to implement Client/Server application using Remote Method Invocation (RMI) to perform arithmetic operations.

Pre- Requisite : Client-server, Object

Theory:

1. RPC: -

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports. RPC makes the client/server model of computing more powerful and easier to program.

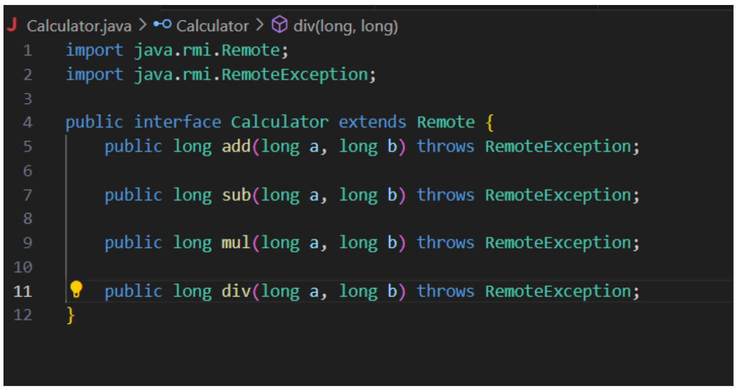
2. RMI: -

RMI (Remote Method Invocation) is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network. RMI is the Java version of generally known as a remote procedure call (RPC), but with the ability to pass one or more objects along with the request. The object can include information that will change the service that is performed in the remote computer. Sun Microsystems, the inventors of Java, calls this "moving behaviour." For example, when a user at a remote computer fills out an expense account, the Java program interacting with the user could communicate, using RMI, with a Java program in another computer that always had the latest policy about expense reporting. In reply, that program would send back an object and associated method information that would enable the remote computer program to screen the user's expense account data in a way that was consistent with the latest policy. The user and the company both would save time by catching mistakes early. Whenever the company policy changed, it would require a change to a program in only one computer. Sun calls its object parameter-passing mechanism object serialization. An RMI request is a request to invoke the method of a remote object. The request has the same syntax as a request to invoke an object method in the same (local) computer. In general, RMI is designed to preserve the object model and its advantages across a network. RMI is implemented as three layers:

- A stub program in the client side of the client/server relationship, and a corresponding skeleton at the server end.
- A Remote Reference Layer that can behave differently depending on the parameters passed by the calling program.
- A Transport Connection Layer, which sets up and manages the request. A single request travels down through the layers on one computer and up through the layers at the other end.

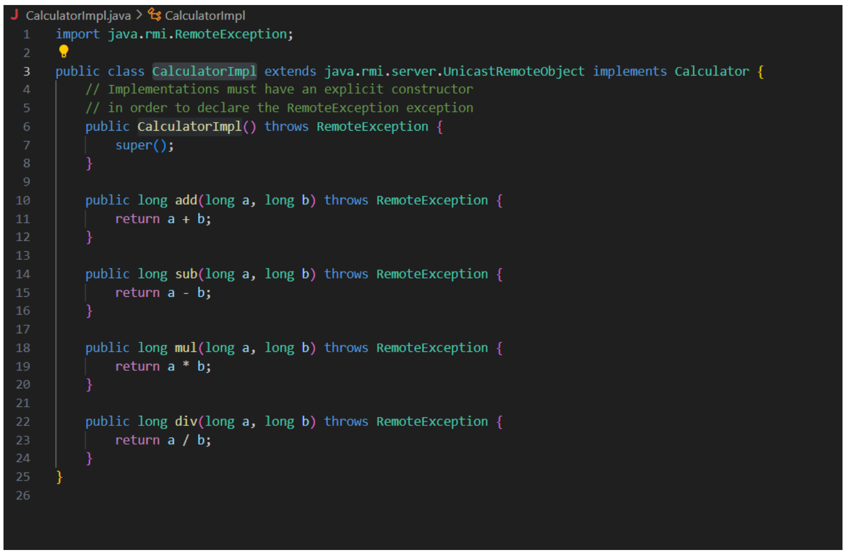
Code:

RMI Interface: Calculator.java(Interface)



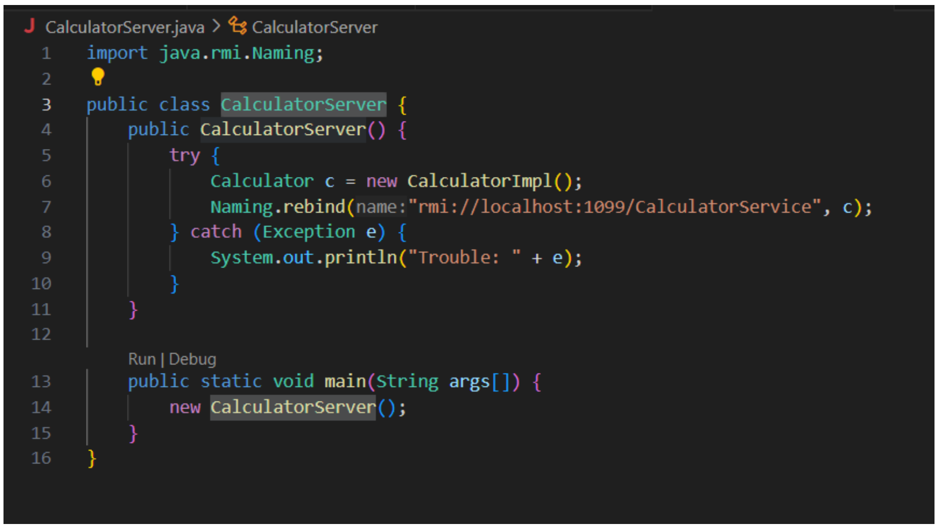
```java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calculator extends Remote {
    public long add(long a, long b) throws RemoteException;

    public long sub(long a, long b) throws RemoteException;

    public long mul(long a, long b) throws RemoteException;

    public long div(long a, long b) throws RemoteException;
}
```

CalculatorImpl.java



```java
import java.rmi.RemoteException;

public class CalculatorImpl extends java.rmi.server.UnicastRemoteObject implements Calculator {
    // Implementations must have an explicit constructor
    // in order to declare the RemoteException exception
    public CalculatorImpl() throws RemoteException {
        super();
    }

    public long add(long a, long b) throws RemoteException {
        return a + b;
    }

    public long sub(long a, long b) throws RemoteException {
        return a - b;
    }

    public long mul(long a, long b) throws RemoteException {
        return a * b;
    }

    public long div(long a, long b) throws RemoteException {
        return a / b;
    }
}

```

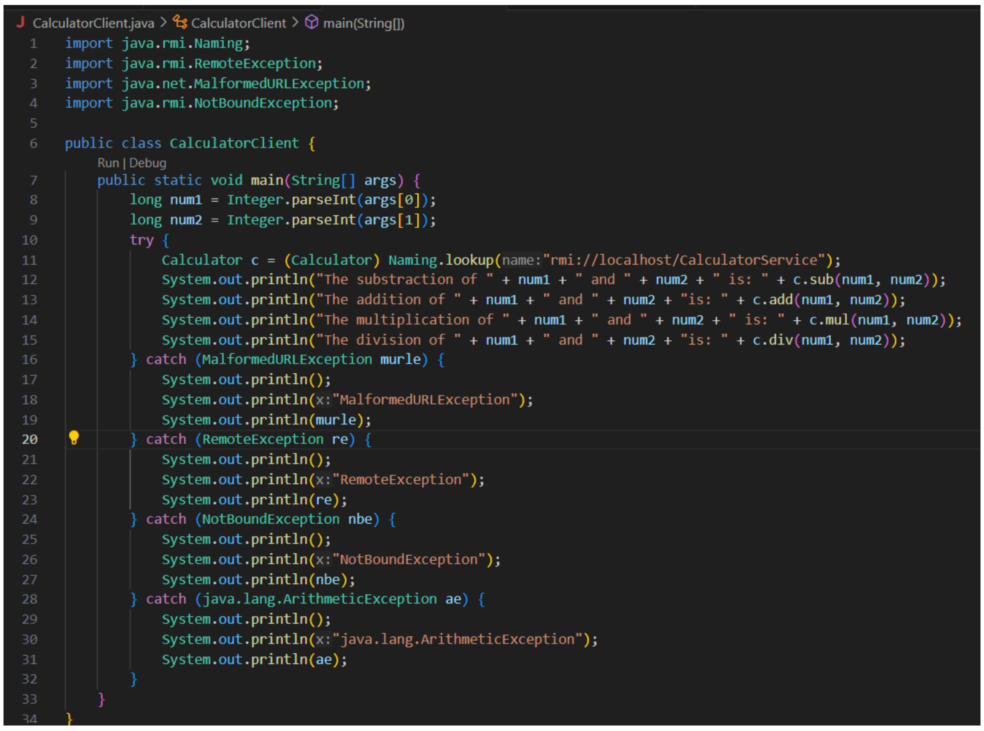
Server: CalculatorServer.java



CalculatorServer.java > CalculatorServer

```java
import java.rmi.Naming;

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

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

CalculatorClient.java



CalculatorClient.java > CalculatorClient > main(String[])

```java
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;

public class CalculatorClient {
    Run | Debug
    public static void main(String[] args) {
        long num1 = Integer.parseInt(args[0]);
        long num2 = Integer.parseInt(args[1]);
        try {
            Calculator c = (Calculator) Naming.lookup(name:"rmi://localhost/CalculatorService");
            System.out.println("The substraction of " + num1 + " and " + num2 + " is: " + c.sub(num1, num2));
            System.out.println("The addition of " + num1 + " and " + num2 + "is: " + c.add(num1, num2));
            System.out.println("The multiplication of " + num1 + " and " + num2 + " is: " + c.mul(num1, num2));
            System.out.println("The division of " + num1 + " and " + num2 + "is: " + c.div(num1, num2));
        } catch (MalformedURLException murle) {
            System.out.println();
            System.out.println(x:"MalformedURLException");
            System.out.println(murle);
        } catch (RemoteException re) {
            System.out.println();
            System.out.println(x:"RemoteException");
            System.out.println(re);
        } catch (NotBoundException nbe) {
            System.out.println();
            System.out.println(x:"NotBoundException");
            System.out.println(nbe);
        } catch (java.lang.ArithmeticException ae) {
            System.out.println();
            System.out.println(x:"java.lang.ArithmeticException");
            System.out.println(ae);
        }
    }
}
```

Execution Steps:

1. Enter and Compile the Source Code
2. Generate Stubs and Skeletons:

   To generate stubs and skeletons, you use a tool called the RMI compiler, which is invoked from the command line, as shown here:

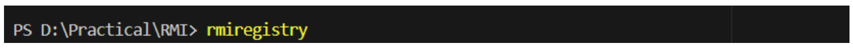
   rmic AddServerImpl or

   javac -h . YourRemoteInterface.java
3. Install Files on the Client and Server Machines

4. Start the RMI Registry on the Server Machine
5. Start the Server
6. Start the Client

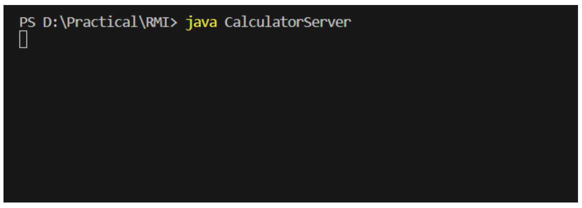
Output:

1. RMI Registry:



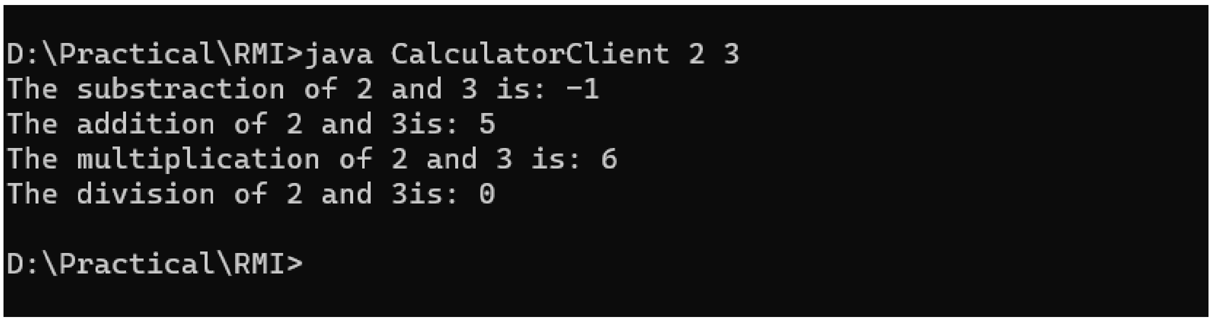
```
PS D:\Practical\RMI> rmiregistry
```

2. Server Side:



```
PS D:\Practical\RMI> java CalculatorServer
```

3. Client Side –



```
D:\Practical\RMI>java CalculatorClient 2 3
The substraction of 2 and 3 is: -1
The addition of 2 and 3is: 5
The multiplication of 2 and 3 is: 6
The division of 2 and 3is: 0

D:\Practical\RMI>
```

Conclusion:

Hence, we have studied and run Client-Server based RPC program successfully and RMI program for performing arithmetic operations such as addition, subtraction, multiplication and division is created and the output is verified.