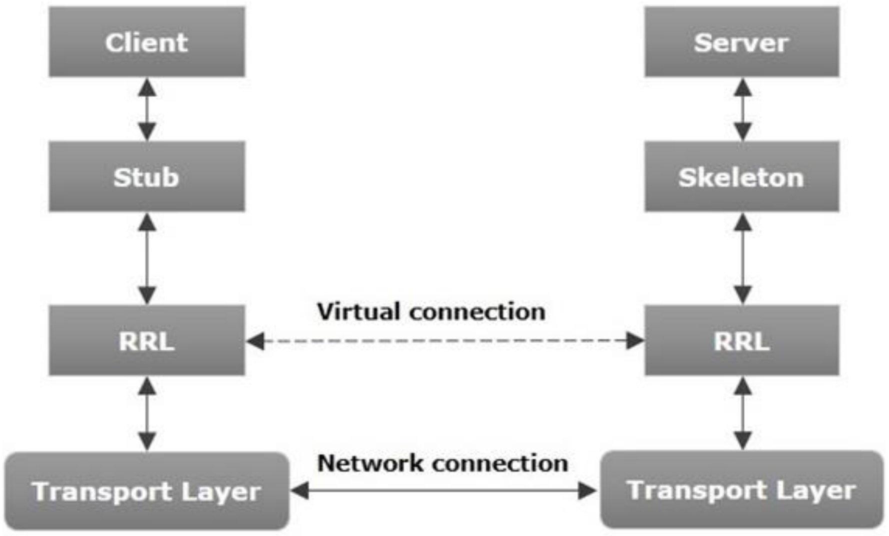
EXPERIMENT 02

CLASS: BE CMPN A 2 **ROLL NO. : 18**
NAME: REBECCA DIAS **PID: 182027**

Aim: Client Server Implementation using RPC/ RMI

Theory:

Remote Method Invocation (RMI)
The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed applications in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton.



Transport Layer − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
Stub − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
Skeleton − This is the object which resides on the server side. stub communicates with this skeleton to pass requests to the remote object.
RRL(Remote Reference Layer) − It is the layer which manages the references made by the client to the remote object.
Server: An program that provides services to objects residing on other machines.
Service contract: An interface that defines the services provided by the server.
Client: An object that uses the services provided by the server program on a remote machine.
Registry: A service provided to clients to locate a server on a remote machine providing the service.

The client and server classes are written by programmers. The stubs and skeletons are created by the RMI compiler from the server code. This is how the RMI process works:

1. The client invokes a method that is offered by the server. The invocation is received by the stub.
2. The stub arranges the request and data in a linear stream. This is known as marshalling and sends that information to the machine that the server resides on.
3. The skeleton receives the request and unmarshall the information and invokes the method on the server object.
4. The server executes the method and returns the result to the skeleton.
5. The skeleton marshals the result and sends it to the stub.
6. The stub unmarshals the result and sends the result to the client object.

In this simple example on Remote Method Invocation, the server offers the services of a calculator that can perform elementary arithmetic operations. These are the steps that you would need to perform to set up the system. The sample code is appended below.

1. Clients must know what services the Calculator Server provides.
   RMI: Calculator interface lists the methods available to remote clients.
2. Clients must find a Calculator object on the server.
   RMI: The Calculator Server registers an object with a special server, called the rmiregistry, so that clients can look it up by name.
3. Clients must be able to pass arguments to, and invoke a method of the Calculator object located on the server. RMI: A special compiler called rmic creates a stub class for use on the client, and a skeleton class for use on the server. The stub takes the request from the client, passes the arguments to the skeleton which invokes the Calculator methods, and sends the return value back to the stub which passes it to the client.

Goals of RMI:

Following are the goals of RMI −

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

Steps:

On the server machine:

1. Compile CalculatorImpl.java, Calculator.java, and CalculatorServer.java
2. Create the stub and skeleton classes using the command **rmic Calculator**

3. Copy the Calculator_Stub.class to the client site
4. Start the registry using the command **start rmiregistry**
5. Start CalculatorServer using **java CalculatorServer serverHostname**

On the client machine:

1. Compile CalculatorImpl.java and CalculatorClient.java
2. Run CalculatorClient using the command java CalculatorClient serverHostname

Program

Calculator.java

```
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;
}
```

CalculatorImpl.java

```
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 java.rmi.RemoteException {
 super();
 }
 public long add(long a, long b)
 throws java.rmi.RemoteException {
 System.out.println ("Doing addition");
 return a + b;
 }
 public long sub(long a, long b)
 throws java.rmi.RemoteException {
 System.out.println ("Doing subtraction");
 return a - b;
 }
 public long mul(long a, long b)
 throws java.rmi.RemoteException {
 System.out.println ("Doing multiplication");  return a * b;
 }
 public long div(long a, long b)
 throws java.rmi.RemoteException {
 System.out.println ("Doing division");
 return a / b;
 }
}
```

CalculatorServer.java

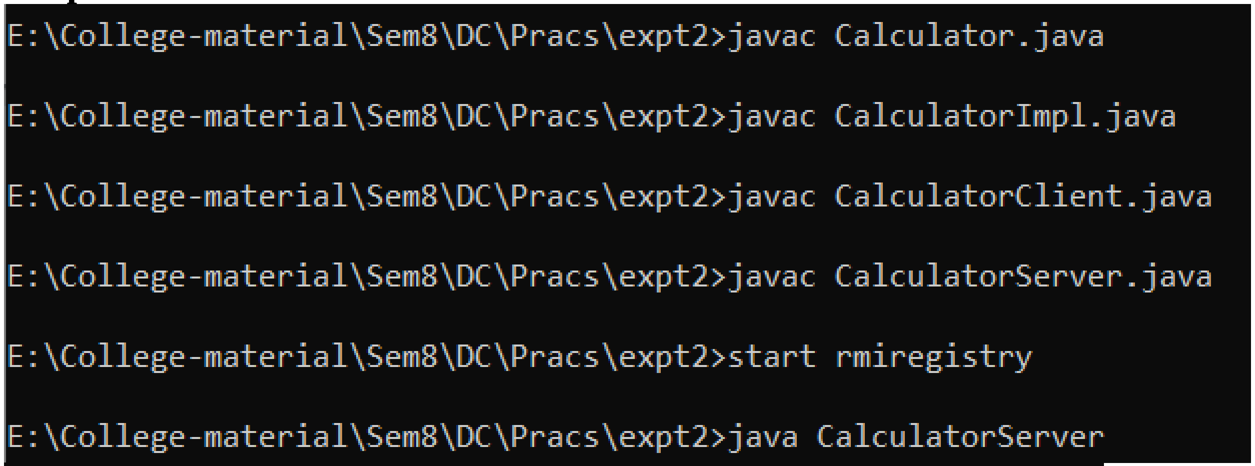
```
import java.rmi.Naming;
public class CalculatorServer {
    public CalculatorServer() {
        try {
            Calculator c = new CalculatorImpl();
            Naming.rebind("rmi://localhost:1099/CalculatorService", c);
        }
        catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }
    public static void main(String args[]) {
        new CalculatorServer();
    }
}
```

CalculatorClient.java

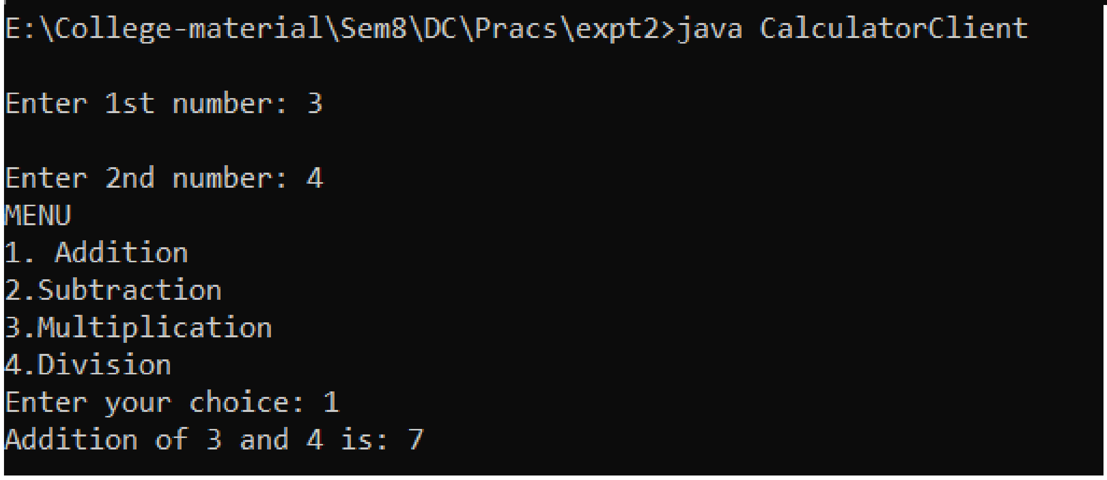
```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.util.Scanner;
import java.net.MalformedURLException; import java.rmi.NotBoundException;
```

```
public class CalculatorClient {
public static void main(String[] args) {
    try {
        Calculator c = (Calculator)
        Naming.lookup("rmi://localhost/CalculatorService");

        int x, y;
        Scanner sc = new Scanner(System.in);

        System.out.print("\nEnter 1st number: ");
        x = sc.nextInt();
        System.out.print("\nEnter 2nd number: ");
        y = sc.nextInt();

        System.out.println("MENU");
        System.out.println("1. Addition \n2.Subtraction \n3.Multiplication \n4.Division");
        System.out.print("Enter your choice: ");

        switch(sc.nextInt()){
            case 1:
                System.out.println("Addition of " + x + " and " + y + " is: " + c.add(x, y));
                break;
            case 2:
                System.out.println("Subtraction of " + x + " and " + y + " is: " + c.sub(x, y));
                break;
            case 3:
                System.out.println("Multiplication of " + x + " and " + y + " is: " + c.mul(x,
y));
                break;
            case 4:
                System.out.println("Division of " + x + " and " + y + " is: " + c.div(x, y));
                break;
        }
    }
    catch (MalformedURLException murle) {
        System.out.println();
        System.out.println("MalformedURLException");
        System.out.println(murle);
    }
    catch (RemoteException re) {
        System.out.println();
        System.out.println("RemoteException");
        System.out.println(re);
    }
    catch (NotBoundException nbe) {  System.out.println();
        System.out.println("NotBoundException");
        System.out.println(nbe);
    }
    catch (
        java.lang.ArithmeticException  ae) {
        System.out.println();
        System.out.println("java.lang.ArithmeticException");
        System.out.println(ae);
    }
 }
}
```

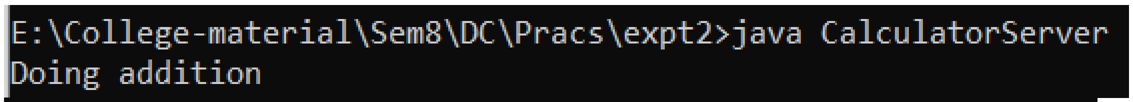
Output:



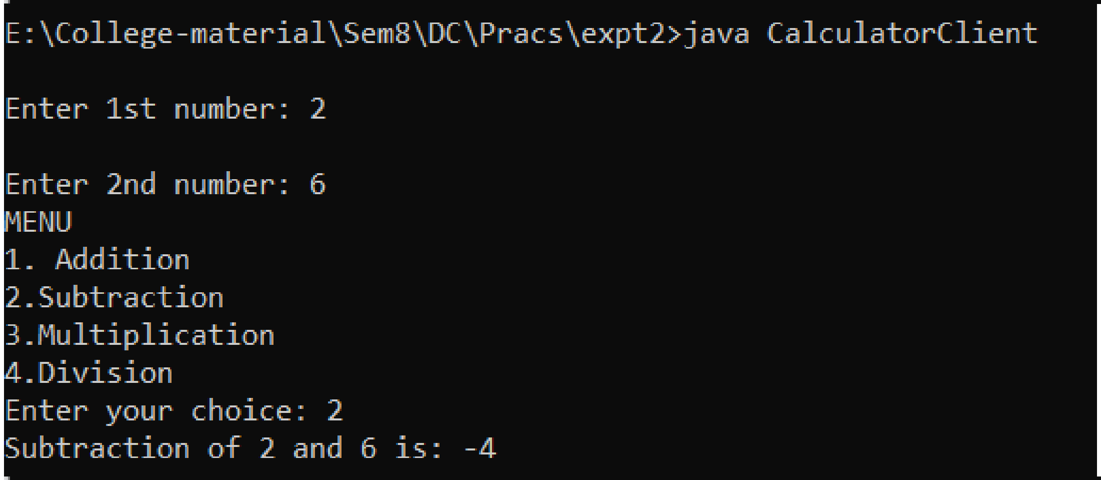
```
E:\College-material\Sem8\DC\Pracs\expt2>javac Calculator.java

E:\College-material\Sem8\DC\Pracs\expt2>javac CalculatorImpl.java

E:\College-material\Sem8\DC\Pracs\expt2>javac CalculatorClient.java

E:\College-material\Sem8\DC\Pracs\expt2>javac CalculatorServer.java

E:\College-material\Sem8\DC\Pracs\expt2>start rmiregistry

E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorServer
```



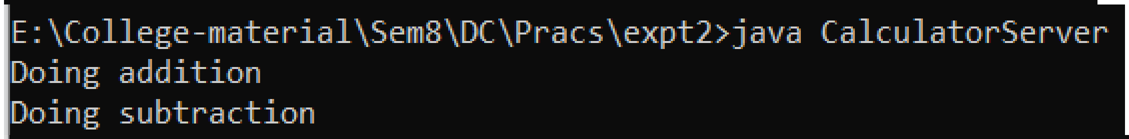
```
E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorClient

Enter 1st number: 3

Enter 2nd number: 4
MENU
1. Addition
2.Subtraction
3.Multiplication
4.Division
Enter your choice: 1
Addition of 3 and 4 is: 7
```



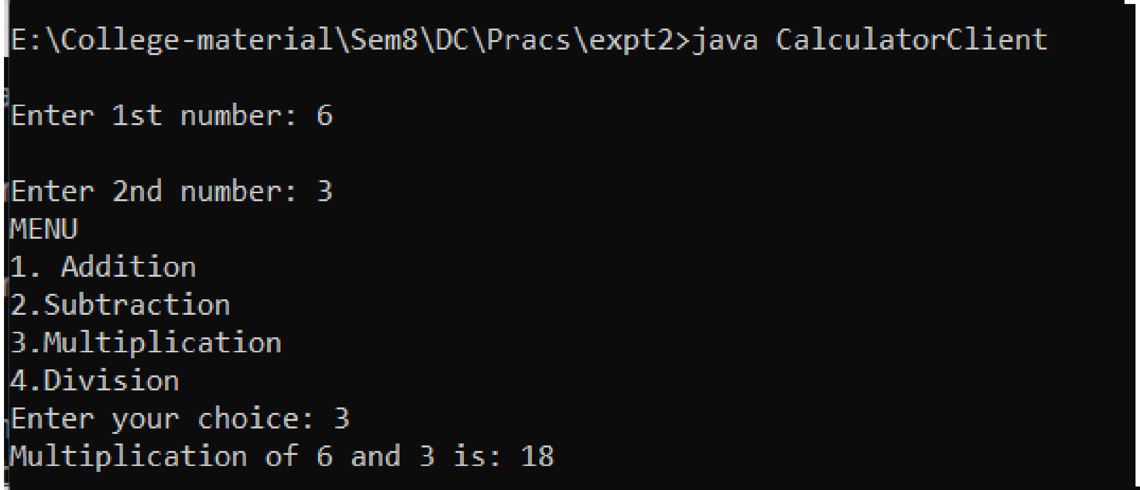
```
E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorServer
Doing addition
```



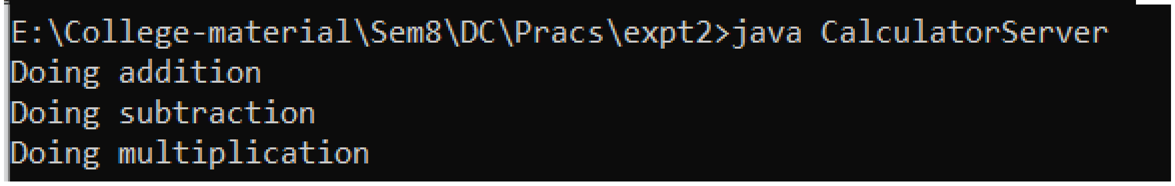
```
E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorClient

Enter 1st number: 2

Enter 2nd number: 6
MENU
1. Addition
2.Subtraction
3.Multiplication
4.Division
Enter your choice: 2
Subtraction of 2 and 6 is: -4
```



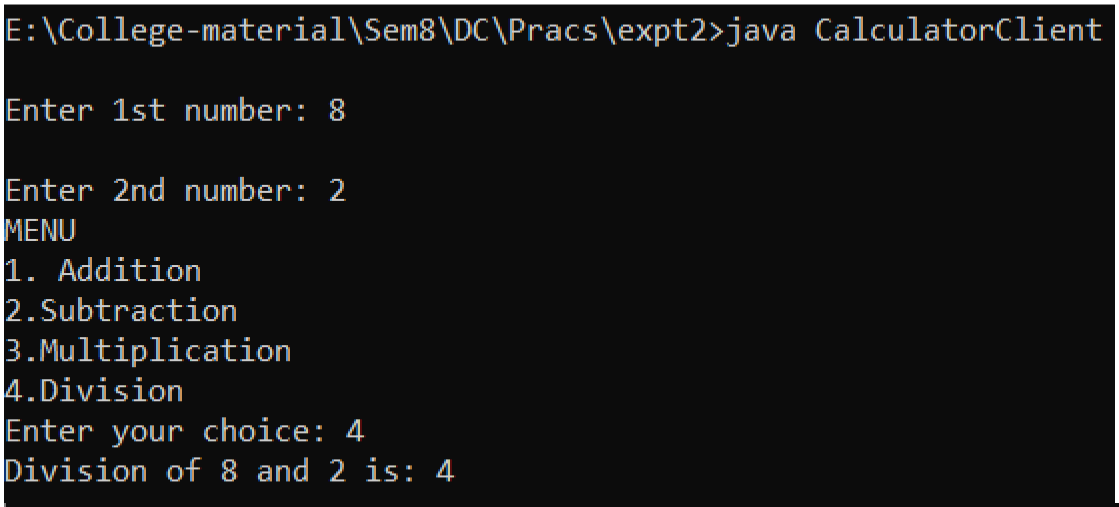
```
E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorServer
Doing addition
Doing subtraction
```



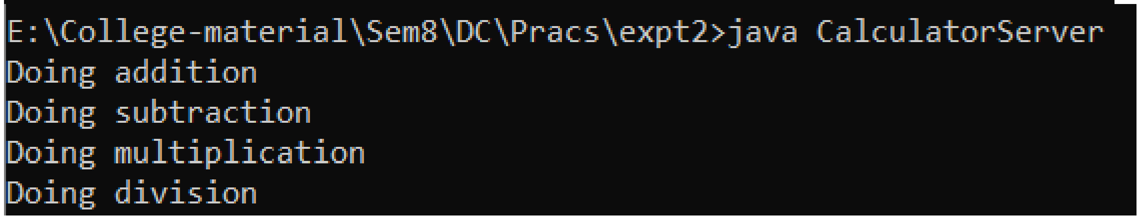
```
E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorClient

Enter 1st number: 6

Enter 2nd number: 3
MENU
1. Addition
2.Subtraction
3.Multiplication
4.Division
Enter your choice: 3
Multiplication of 6 and 3 is: 18
```



```
E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorServer
Doing addition
Doing subtraction
Doing multiplication
```



```
E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorClient

Enter 1st number: 8

Enter 2nd number: 2
MENU
1. Addition
2.Subtraction
3.Multiplication
4.Division
Enter your choice: 4
Division of 8 and 2 is: 4
```



```
E:\College-material\Sem8\DC\Pracs\expt2>java CalculatorServer
Doing addition
Doing subtraction
Doing multiplication
Doing division
```

Conclusion:

In this experiment, we learnt about Remote Method Invocation (RMI) in Java. The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed applications in java. We implemented the Client Server model using rmi wherein the client could perform arithmetic operations like addition, subtraction, multiplication and division. The code was written in Java and the output was obtained successfully.