

/\*

Name: Rohit Saini

Erp: 1032200897

Panel: C

RollNo: PC-41 \*/

**TITLE :** Design a distributed application using RMI.

**AIM :** To implement a basic calculator ( operations : +, - , /, \*)using RMI.

**OBJECTIVE :** To implement RMI for implementing basic mathematical operations.

To have a client and server execute the RMI.

**THEORY:**

**RMI (Remote Method Invocation):**

RMI, or Remote Method Invocation, is a Java-based technology that enables the execution of methods on objects residing in a different Java Virtual Machine (JVM) across a network. It simplifies the development of distributed applications in Java by allowing objects in one JVM to invoke methods on objects in another JVM.

**Components of RMI:**

1. **Remote Interface:**
   * Defines methods that can be invoked remotely.
   * Methods must throw **RemoteException** to indicate they can be called remotely.
2. **Remote Object:**
   * Implements the remote interface.
   * Extends **UnicastRemoteObject** or uses **exportObject** for remote accessibility.
   * Handles the actual implementation of remote methods.
3. **Registry:**
   * Acts as a lookup service for remote objects.
   * Server registers its remote object with the registry.
   * Clients look up remote objects using the registry.

**Two Types of Remote Classes in RMI:**

1. **Extending UnicastRemoteObject:**
   * Remote class extends **UnicastRemoteObject**.
   * Convenient for making a class remotely accessible.
2. **Using exportObject Method:**

* Remote class implements the remote interface.
* Uses **UnicastRemoteObject.exportObject** method for remote accessibility.

**PSEUDO CODE/STEPS OF ALGORITHM :**

RemoteInterface:

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface MyRemoteInterface extends Remote {

    int add(int a, int b) throws RemoteException;

    int subtract(int a, int b) throws RemoteException;

    int multiply(int a, int b) throws RemoteException;

    double divide(int a, int b) throws RemoteException;

}

MyServer:

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

public class MyServer extends UnicastRemoteObject implements MyRemoteInterface {

    public MyServer() throws RemoteException {

        // Constructor to declare that it throws RemoteException

    }

    public int add(int a, int b) throws RemoteException {

        return a + b;

    }

    public int subtract(int a, int b) throws RemoteException {

        return a - b;

    }

    public int multiply(int a, int b) throws RemoteException {

        return a \* b;

    }

    public double divide(int a, int b) throws RemoteException {

        if (b == 0) {

            throw new RemoteException("Division by zero is not allowed.");

        }

        return (double) a / b;

    }

    public static void main(String[] args) {

        try {

            MyServer server = new MyServer();

            java.rmi.Naming.rebind("MyServer", server);

            System.out.println("Calculator Server is running...");

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

MyClient:

import java.rmi.Naming;

public class MyClient {

    public static void main(String[] args) {

        try {

            MyRemoteInterface remoteObject = (MyRemoteInterface) Naming.lookup("rmi://localhost/MyServer");

            /\* reading file for reading value of a and b \*/

            java.io.File file = new java.io.File("input.txt");

            java.util.Scanner input = new java.util.Scanner(file);

            int a = input.nextInt();

            int b = input.nextInt();

            input.close();

            System.out.println("Server says: " + a + " + " + b + " = " + remoteObject.add(a, b));

            System.out.println("Server says: " + a + " - " + b + " = " + remoteObject.subtract(a, b));

            System.out.println("Server says: " + a + " \* " + b + " = " + remoteObject.multiply(a, b));

            System.out.println("Server says: " + a + " / " + b + " = " + remoteObject.divide(a, b));

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

**PLATFORM :** Linux

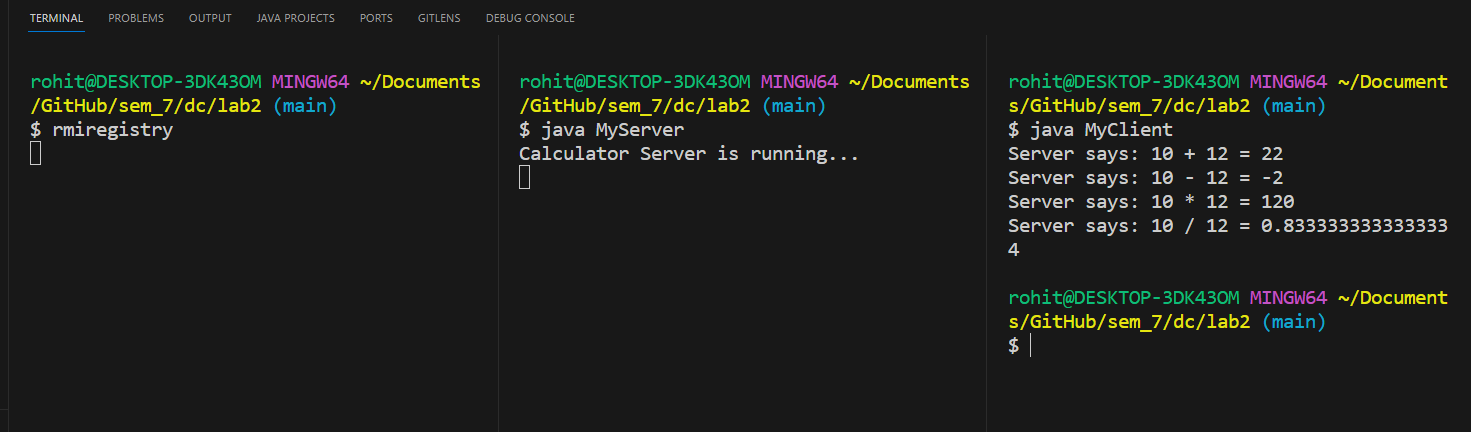
**PROGRAMMING LANGUAGE:** C,compiler :gcc/cc

**PLATFORM:**  Linux

**LANGUAGE:** C

**INPUT:**  The values for mathematical operations

**OUTPUT:** The output based on +, - ,/ ,\* operations



**CONCLUSION :** Thus, RMI has been studied and implemented on Linux platform.

**FAQs :** 1. Differentiate between RPC and RMI

2. what does rmic do ?

3.what is the importance of RMI registry

