| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Practical Number:** 5 | |
| **Aim of Practical:**  Implementation of equation solver using Remote Method Invocation (RMI). | | | |
| **DOP:** 14.9.23 | | **DOS:** 22.10.23 | |
| **CO Mapped:** CO1 | **PO Mapped:** - | **Faculty Signature:** | **Marks:** |

**Aim: Implementation of equation solver using Remote Method Invocation (RMI).**

**Description:**

Equation solver. 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 = a2 –2ab + b2;

If a = 5 and b = 2 then return value = 52 – 2.5.2 + 22 = 9

**RMI (Remote Method Invocation):**

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application 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. RMI uses stub and skeleton object for communication with the remote object. A remote object is an object whose method can be invoked from another JVM.

**Stub:**

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),

2. It writes and transmits (marshals) the parameters to the remote Virtual Machine(JVM),

3. It waits for the result

4. It reads (unmarshals) the return value or exception, and

5. It finally, returns the value to the caller.

**Skeleton:**

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method

2. It invokes the method on the actual remote object, and

3. It writes and transmits (marshals) the result to the caller.

The 6 steps to write

1. Create the remote interface

2. Provide the implementation of the remote interface

3. Compile the implementation class and create the stub and skeleton objects using the rmic tool

4. Start the registry service by rmiregistry tool

5. Create and start the remote application

6. Create and start the client application

**SOURCE CODE:**

**EqualtionSolver.java:**

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.ArrayList;

@FunctionalInterface

public interface EquationSolver extends Remote {

int evaluateEquation(String equation, ArrayList<Integer> params) throws RemoteException;

}

**EquationSolverImpl.java:**

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

import java.util.ArrayList;

import java.util.HashMap;

public class EquationSolverImpl extends UnicastRemoteObject implements EquationSolver {

private static final long serialVersionUID = 1L;

EquationSolverImpl() throws RemoteException {

super();

}

@Override

public int evaluateEquation(String equation, ArrayList<Integer> params) {

System.out.print("Equation recieved: " + equation);

int a = 0, b = 0, c = 0;

HashMap<String, Integer> map = new HashMap<>();

if (params.size() == 2) {

params.add(c);

}

map.put("a^2-b^2", (params.get(0) - params.get(1)) \* (params.get(0) + params.get(1)));

map.put("a^2+b^2", (int) (Math.pow((params.get(0) - params.get(1)), 2) + 2 \* params.get(0) \* params.get(1)));

map.put("(a+b)^2",

(int) (Math.pow(params.get(0), 2) + 2 \* params.get(0) \* params.get(1) + Math.pow(params.get(1), 2)));

map.put("(a-b)^2",

(int) (Math.pow(params.get(0), 2) - 2 \* params.get(0) \* params.get(1) + Math.pow(params.get(1), 2)));

map.put("(a+b+c)^2",

(int) (Math.pow(params.get(0), 2) + Math.pow(params.get(1), 2) + Math.pow(params.get(2), 2)

+ 2 \* params.get(0) \* params.get(1) + 2 \* params.get(1) \* params.get(2)

+ 2 \* params.get(0) \* params.get(2)));

map.put("(a-b-c)^2",

(int) (Math.pow(params.get(0), 2) + Math.pow(params.get(1), 2) + Math.pow(params.get(2), 2)

- 2 \* params.get(0) \* params.get(1) + 2 \* params.get(1) \* params.get(2)

- 2 \* params.get(0) \* params.get(1)));

map.put("a^3-b^3", (int) ((params.get(0) - params.get(1))

\* (Math.pow(params.get(0), 2) + params.get(0) \* params.get(1) + Math.pow(params.get(1), 2))));

map.put("a^3+b^3", (int) ((params.get(0) + params.get(1))

\* (Math.pow(params.get(0), 2) - params.get(0) \* params.get(1) + Math.pow(params.get(1), 2))));

map.put("(a+b)^3", (int) (Math.pow(params.get(0), 3) + 3 \* Math.pow(params.get(0), 2) \* params.get(1)

+ 3 \* params.get(0) \* Math.pow(params.get(1), 2) + Math.pow(params.get(1), 3)));

map.put("(a-b)^3", (int) (Math.pow(params.get(0), 3) - 3 \* Math.pow(params.get(0), 2) \* params.get(1)

+ 3 \* params.get(0) \* Math.pow(params.get(1), 2) - Math.pow(params.get(1), 3)));

int answer = map.get(equation);

System.out.print(answer);

return answer;

}

}

**MyServer.java:**

import java.rmi.Naming;

public class MyServer {

public static void main(String args[]) {

System.out.print("Hello");

try {

System.out.print("Server started, waiting for client to enter equation");

EquationSolver stub = new EquationSolverImpl();

Naming.rebind("rmi://localhost:5000/ajayk57EquationSolver", stub);

} catch (Exception e) {

System.out.print("exception on server");

System.out.println(e);

}

}

}

**MyClient.java:**

import java.rmi.Naming;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.Scanner; // Import the Scanner class

import java.util.Set;

import java.util.TreeSet;

public class MyClient {

public static void main(String args[]) {

try (Scanner sc = new Scanner(System.in)) {

while (true) {

try {

System.out.print("Enter equation: ");

String equation = sc.nextLine();

Set<String> tree = new TreeSet<>();

ArrayList<Integer> params = new ArrayList<Integer>();

for (char c : equation.toCharArray()) {

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {

tree.add(Character.toString(c));

}

}

Iterator<String> value = tree.iterator();

// tree.forEach(ele -> {

// params.add(sc.nextInt());

// });

while (value.hasNext()) {

System.out.println("Enter value for " + value.next());

int temp = sc.nextInt();

params.add(temp);

}

EquationSolver stub = (EquationSolver) Naming.lookup("rmi://localhost:5000/ajayk57EquationSolver");

System.out.println("Answer: " + stub.evaluateEquation(equation, params));

} catch (Exception e) {

System.out.println(e);

}

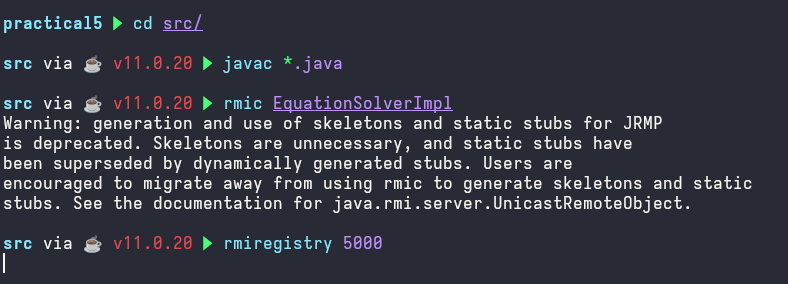
}

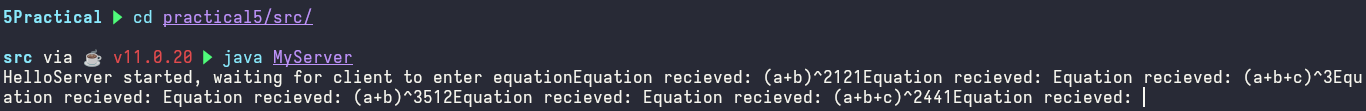
}

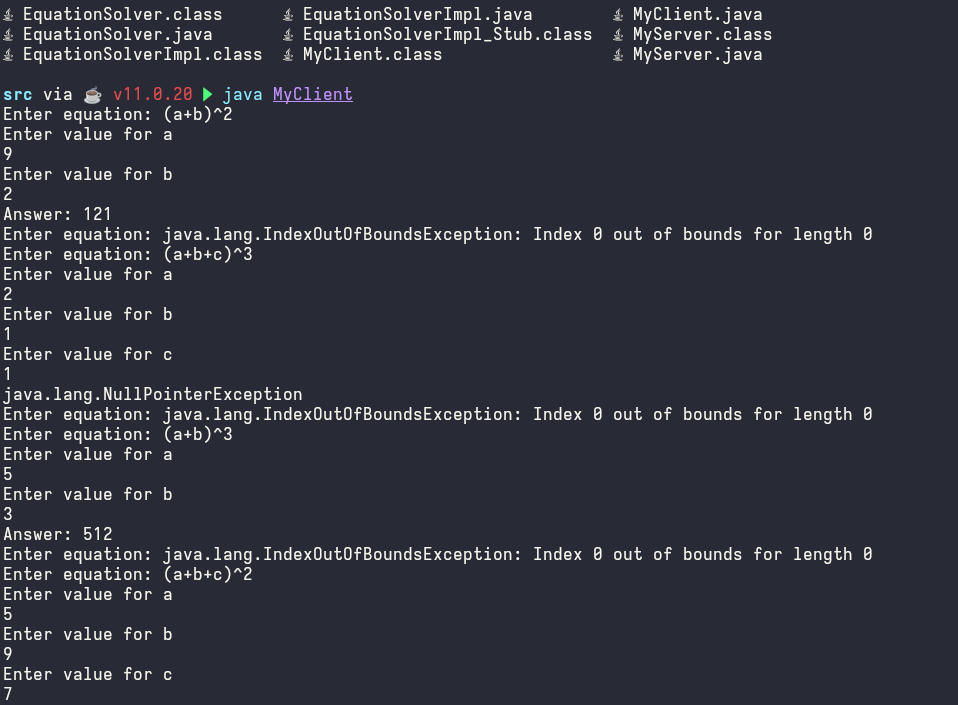
}

}

**OUTPUT:**







**CONCLUSION:**

Hence we successfully implemented equation solver using remote method

invocation