

Laboratory 1

Title of the Laboratory Exercise: RMIServerApp

1. Introduction and Purpose of Experiment

Aim and Objectives

Aim

- To develop a program to perform RMI client server application

2. Experimental Procedure

- Analyse the problem statement
- Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- Implement the algorithm in Java language
- Compile the Java program
- Test the implemented program
- Document the Results

3. Computations/Algorithms

RMI is an API that facilitates to create distributed application in Java. RMI provides remote communication between the applications. In this exercise the addition, subtraction, multiplication and division of two number is computed by the server and the client sends the numbers. An interface is created in which all the required methods are declared and the implementation is done in server class.

Algorithm

STEP 1: START

STEP 2: Create the remote interface which has all methods declared in it.

STEP 3: Create a server class and create a registry which accepts requests on the "server_1".

STEP 4: Provide the implementation of the remote interface

STEP 5: Initialize RMI Registry

STEP 6: Create and start the remote (server) application

STEP 8: Create and start the client application

STEP 9: Using lookup method see for the specified service. Ask the user to give input. Get result from server and print.

STEP 10: Print if any exception is caught in try block.

STEP 11: STOP.

4. Presentation of Results

Interface Source code

```
import java.rmi.*;

public interface InterfaceServerFunc extends Remote {
    public double sum(double a, double b) throws RemoteException;

    public double subtract(double a, double b) throws RemoteException;

    public double multiply(double a, double b) throws RemoteException;

    public double divide(double a, double b) throws RemoteException;
}
```

Server Functions Source Code

```
import java.rmi.*;
import java.rmi.server.*;

public class ServerFunc extends UnicastRemoteObject implements InterfaceServerFunc {

    public ServerFunc() throws RemoteException {
        System.out.println("New Server Initiated...");
    }
    public double sum(double a, double b) throws RemoteException {
        return a + b;
    }
    public double subtract(double a, double b) throws RemoteException {
        return a - b;
    }
    public double multiply(double a, double b) throws RemoteException {
        return a * b;
    }
    public double divide(double a, double b) throws RemoteException {
        return a / b;
    }
}
```

Server Main Source Code

```
import java.rmi.*;

public class Server {
    public static void main(String argv[]) {
        try {
            System.out.println("Starting Server");

            Naming.rebind("server_1", new ServerFunc());

        } catch (Exception e) {
            System.out.println("Problem occurred while starting the server\n"+e.toString());
        }
    }
}
```

Client Main Source Code

```
import java.rmi.*;
import java.util.Scanner;

public class Client {
    public Client() {

        System.out.println("Starting the client");

        // trying to access the Registry server to retrieve the interface
        try {
            msi = (InterfaceServerFunc) Naming.lookup("rmi://127.0.0.1/server_1");
        } catch (Exception e) {
            System.out.println("Client boot Failed\n" + e);
            System.out.println("Make sure that both the Registry server and the server
application are running correctly");

            System.exit(0);
        }
    }

    public static void main(String[] argv) {

        Client c = new Client();
        try {

            Scanner sc = new Scanner(System.in);
            int a, b;

            System.out.print("Enter first operand : ");
            a = sc.nextInt();
            System.out.print("Enter second operand : ");
            b = sc.nextInt();

            System.out.println("Add: " + c.msi.sum(a, b));
            System.out.println("Subtract: " + c.msi.subtract(a, b));
            System.out.println("Multiply: " + c.msi.multiply(a, b));
            System.out.println("Divide: " + c.msi.divide(a, b));

            sc.close();

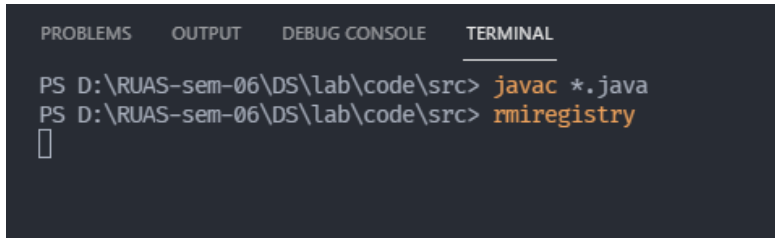
        } catch (Exception e) {
            System.out.println("Error occurred during remote calls : " + e);
        }

    }

    private InterfaceServerFunc msi; // A interface to the remote object
}
```

Results

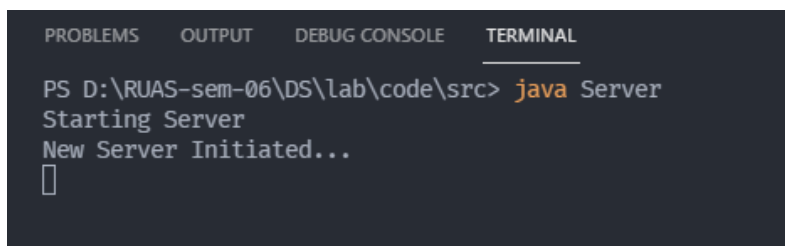
Registry Server



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS D:\RUAS-sem-06\DS\lab\code\src> javac *.java
PS D:\RUAS-sem-06\DS\lab\code\src> rmiregistry
█
```

Figure 1 Output of running rmi_registry

Server side

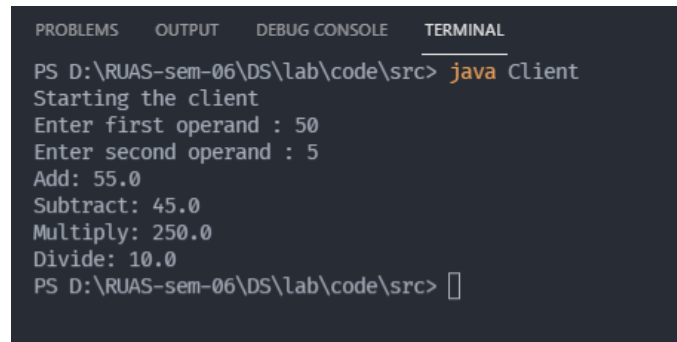


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS D:\RUAS-sem-06\DS\lab\code\src> java Server
Starting Server
New Server Initiated...
█
```

Figure 2 Output of Server Side

Client side

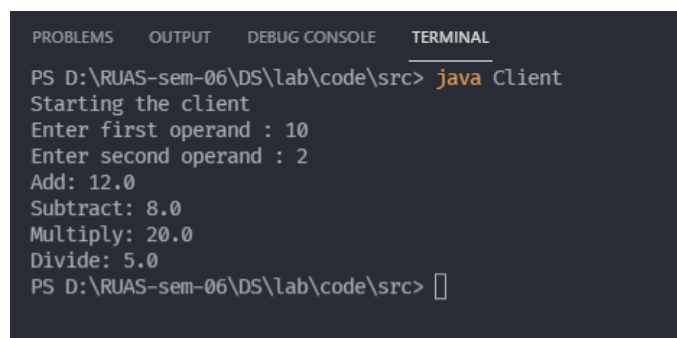
Test 1:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS D:\RUAS-sem-06\DS\lab\code\src> java Client
Starting the client
Enter first operand : 50
Enter second operand : 5
Add: 55.0
Subtract: 45.0
Multiply: 250.0
Divide: 10.0
PS D:\RUAS-sem-06\DS\lab\code\src> █
```

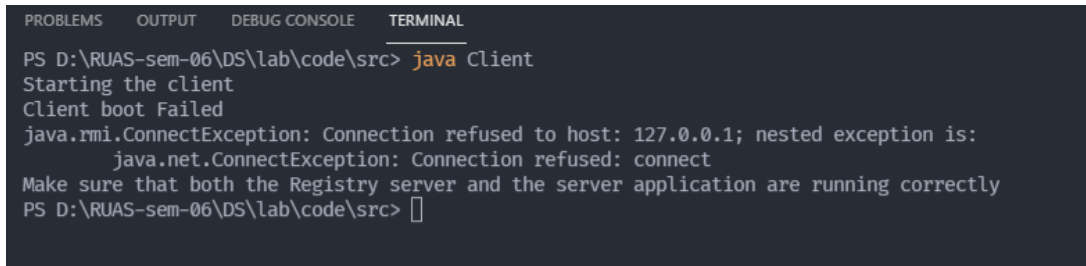
Figure 3 Output of Client-Side test 1

Test 2:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS D:\RUAS-sem-06\DS\lab\code\src> java Client
Starting the client
Enter first operand : 10
Enter second operand : 2
Add: 12.0
Subtract: 8.0
Multiply: 20.0
Divide: 5.0
PS D:\RUAS-sem-06\DS\lab\code\src> █
```

Figure 4 Output of Client-Side test 2

Testing error when client is executed without running the server or registry server.A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' being the active tab. The terminal shows the following text:

```
PS D:\RUAS-sem-06\DS\lab\code\src> java Client
Starting the client
Client boot Failed
java.rmi.ConnectException: Connection refused to host: 127.0.0.1; nested exception is:
    java.net.ConnectException: Connection refused: connect
Make sure that both the Registry server and the server application are running correctly
PS D:\RUAS-sem-06\DS\lab\code\src> 
```

Figure 5 Exception caught Error

5. Analysis and Discussions**a) Limitations of Experiments:**

- RMI is less efficient than Socket objects.
- We cannot use the RMI code out of the scope of java.
- Security issues need to be monitored more closely.

b) Learning happened:

In this program we learned to associate an object in one system (JVM) to an object running on another object. RMI is used to build distributed applications; it provides remote communication between java programs. It is provided in the package *java.rmi*.

Java.rmi.Naming class contains a method to bind, unbind or rebind names with a remote object present at the remote registry. This class is also used to get the reference of the object present at remote registries or the list of names associated with this registry.

c) Recommendations: none**6. Conclusion:**

In this session RMI was used to run a simple Calculator operation like addition, subtraction, multiplication and division. In this program client sends two numbers and server performs operation on the numbers and returns the results.