Assignment 1 B

Roll no: 43144

Name: Ruchika Pande

Title: Any distributed application using RMI technique

Problem Statement:

> To develop a simple calculator through implementing client-server communication programs based on RMI techniques.

Objectives:

- To study about client-server communication
- To study RMI technique

Theory:

RMI (Remote Method Invocation)

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*.

Stub and Skeleton:

RMI uses stub and skeleton objects for communication with the remote object.

A remote object is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

Stub:

The stub is an object, and 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, and 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:

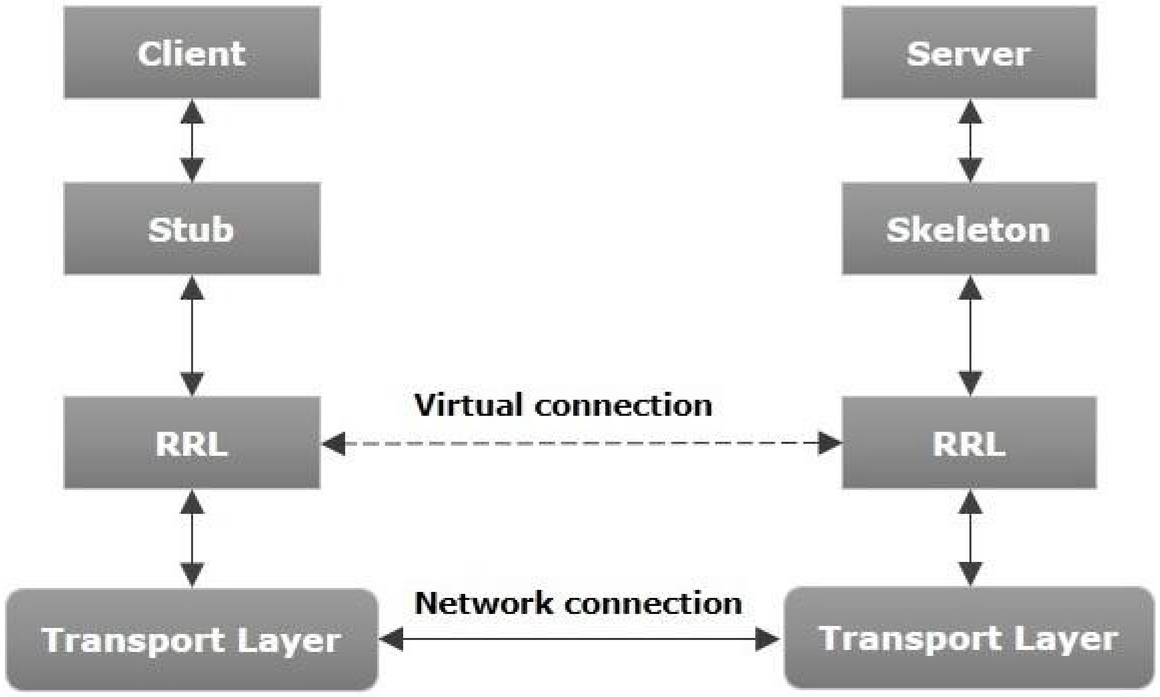
1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

Architecture of RMI Application:

In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).

- Inside the server program, a remote object is created and reference to that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.



Let us now discuss the components of this architecture.

- 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 request to the remote object.
- RRL(Remote Reference Layer) − It is the layer which manages the references made by the client to the remote object.

Working of an RMI Application:

The following points summarize how an RMI application works −

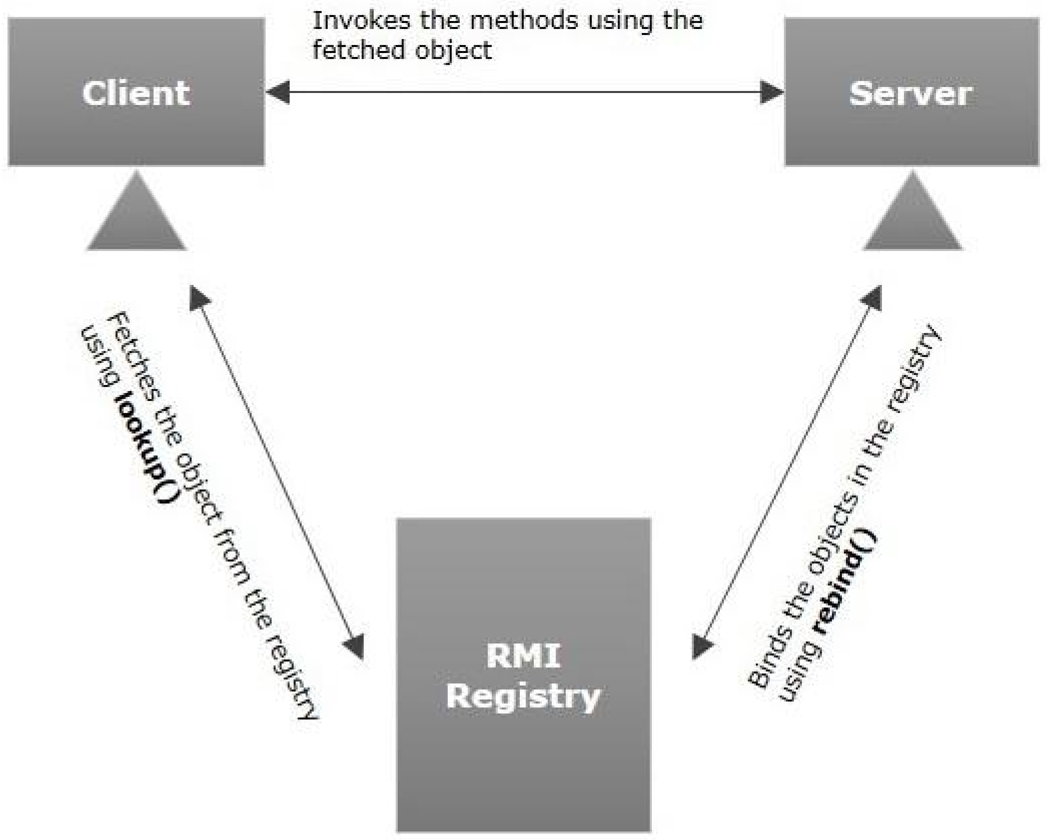
- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called invoke() of the object remoteRef. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

RMI Registry:

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using bind() or reBind() methods). These are registered using a unique name known as bind name.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using lookup() method).

The following illustration explains the entire process −


Invokes the methods using the fetched object
Client
Server
Fetches the object from the registry using **lookup()**
Binds the objects in the registry using **rebind()**
RMI Registry

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 to write RMI calculator program:

1. Create the remote interface calculator
2. Provide the implementation of the remote interface in another class or in the server program.
3. Create and start the server application
4. Create and start the client application

Code:

Client-Server communication using RMI:

Interface:

```
package rmi;

import java.rmi.*;


public interface calculator extends Remote

{

  int add (int d1, int d2) throws Exception;

  int sub (int d1, int d2) throws Exception;

  int mul (int d1, int d2) throws Exception;

  int div (int d1, int d2) throws Exception;

}
```

Client Side Programming:

```java
package rmi;

import java.rmi.*;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.util.Scanner;

import java.io.*;

public class client {



  public static void main(String[] args) throws RemoteException{

    client c = new client();

    c.connectRemote();


  }


  private void connectRemote() throws RemoteException{

    try {

      Scanner sc = new Scanner(System.in);

      Registry reg = LocateRegistry.getRegistry("localhost", 9998);

      calculator ad = (calculator)reg.lookup("add server");

      System.out.println("Enter the two numbers:");

      int a = sc.nextInt();
```

```
            int b = sc.nextInt();

            System.out.println("Addition is "+ad.add(a, b));

            System.out.println("Multiplication is "+ad.mul(a, b));

            System.out.println("Subtaction is "+ad.sub(a, b));

            System.out.println("Division is "+ad.div(a, b));

        }

        catch(NotBoundException|RemoteException e){

            System.out.println("Exception is" + e);

        } catch (Exception e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

    }

}
```

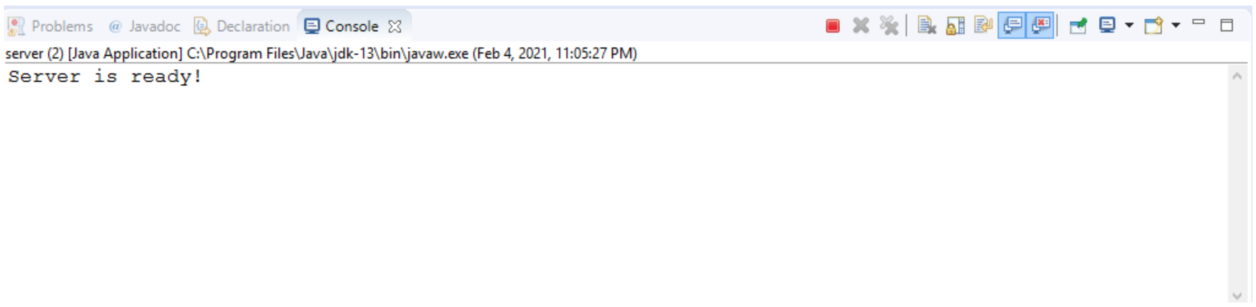
Server Side Programming

```
package rmi;


import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;


public class server extends UnicastRemoteObject

implements calculator{


   protected server() throws RemoteException {

      super();

      // TODO Auto-generated constructor stub

   }


   @Override

   public int add(int d1, int d2) throws Exception {

      // TODO Auto-generated method stub

      return d1+d2;


   }

   @Override
```

```
public int sub(int d1, int d2) throws Exception {
    // TODO Auto-generated method stub
    return d1-d2;
}


@Override
public int mul(int d1, int d2) throws Exception {
    // TODO Auto-generated method stub
    return d1*d2;
}


@Override
public int div(int d1, int d2) throws Exception {
    // TODO Auto-generated method stub
    return d1/d2;
}
public static void main(String[] args) throws RemoteException{
    try {
        Registry reg = LocateRegistry.createRegistry(9998);
        reg.rebind("add server", new server());
        System.out.println("Server is ready!");
    }
    catch(RemoteException e)

    {

      System.out.println("Exception "+ e);

    }

  }

}
```

Output for RMI:

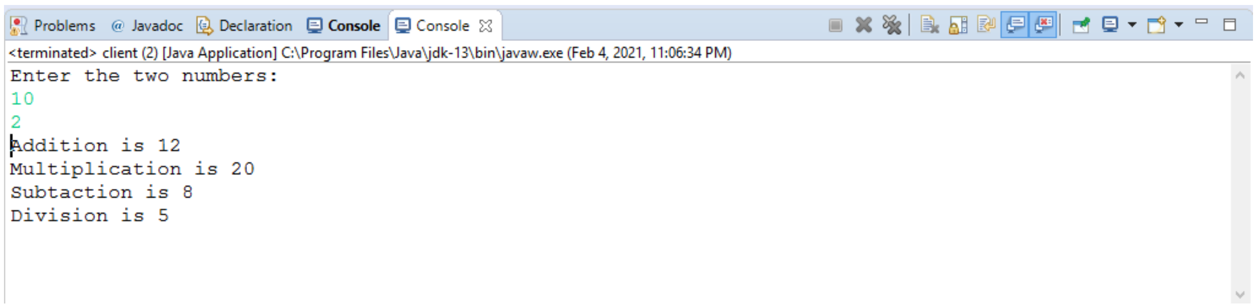
Server Side Terminal:



server (2) [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (Feb 4, 2021, 11:05:27 PM)

```
Server is ready!
```

Client Side Terminal:



<terminated> client (2) [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (Feb 4, 2021, 11:06:34 PM)

```
Enter the two numbers:
10
2
Addition is 12
Multiplication is 20
Subtaction is 8
Division is 5
```

Conclusion:

In this assignment, we learned about client-server communication through different protocols and sockets. We also learned about Java support through the socket API for TCP, UDP programming and RMI techniques.