Experiment 2

Remote Method Innovation Method

Name: Sai Harsha Vardhan AVN

Roll-no: BCSE1823

Batch: ‘A’

Aim: Write a Java program for Remote method Invocation.

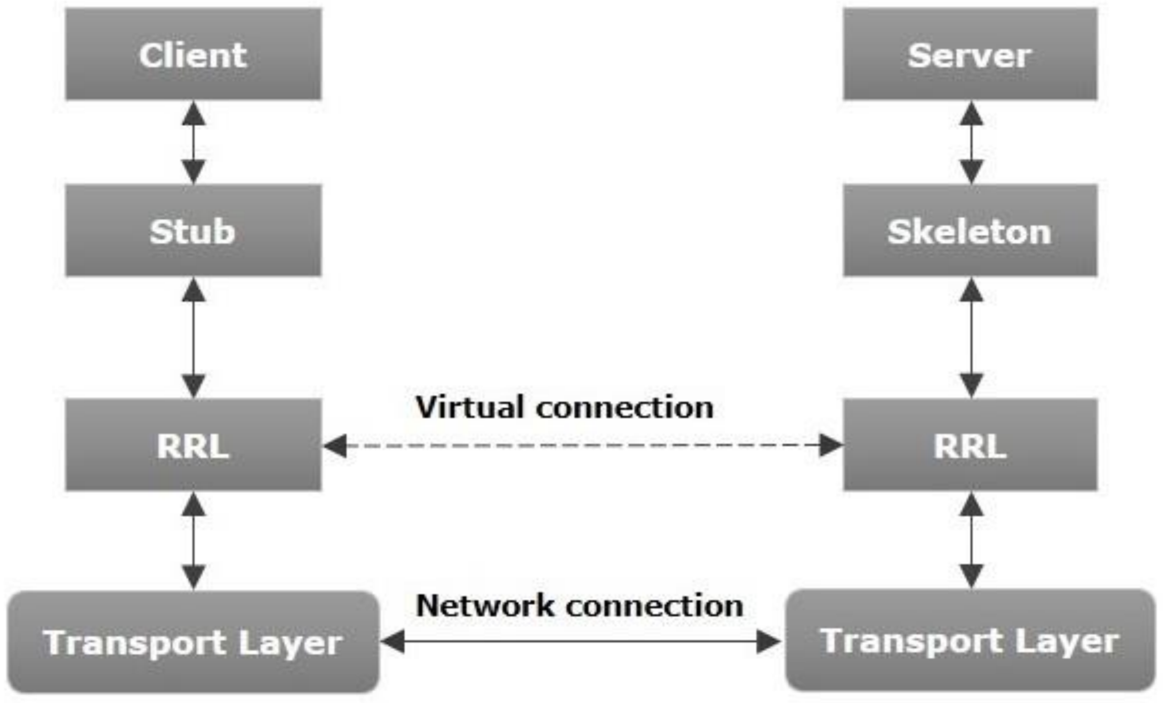
Theory:

RMI stands for **Remote Method Invocation**. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

Architecture of an 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 of 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.



Working of an RMI Application

The following points summarize how an RMI application works −

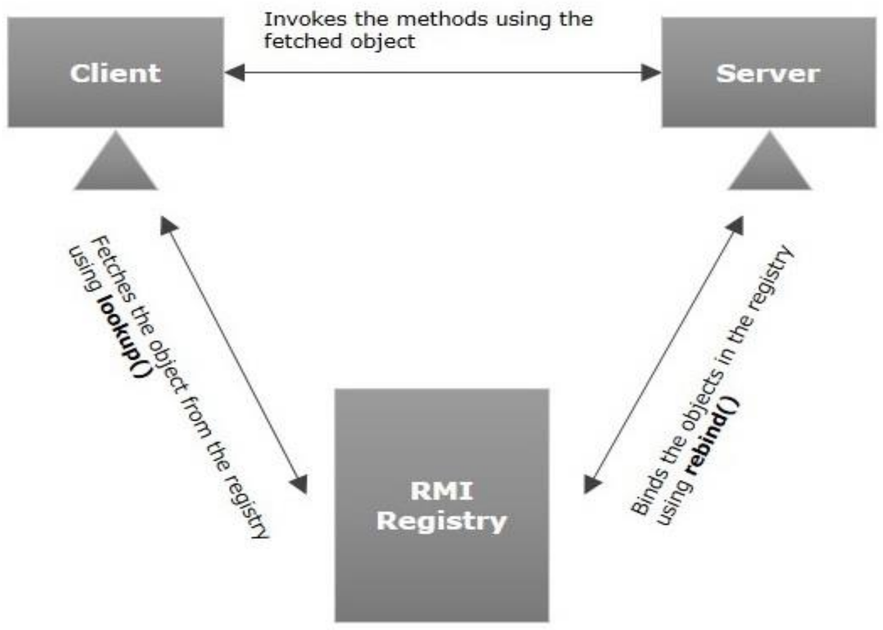
- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.

- 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).


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.
-

Code:

1) ***<u>Message-code</u>***:

```
import java.rmi.*;
import java.rmi.RemoteException;
public interface Hello extends Remote{
   void printMsg() throws RemoteException;
   int adder(int x,int y)throws RemoteException;
}
```

2) ***<u>Server-code</u>***:

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Server extends UnicastRemoteObject implements Hello{

   public Server() throws RemoteException{}
   @Override
   public void printMsg() throws RemoteException {
       System.out.println("This is an example RMI program"); //To change body of
generated methods, choose Tools | Templates.
   }

   @Override
   public int adder(int x, int y) throws RemoteException {
      return(x+y);
   }
   public static void main(String agrgs[]){
      try{
         Registry registry = LocateRegistry.createRegistry(8000);
         registry.rebind("Hello server",new Server());
         System.out.println("Server Ready");
      }catch(Exception ex){
         System.out.println("Server Exception:"+ex.toString());
         ex.printStackTrace();
      }
   }
}
```

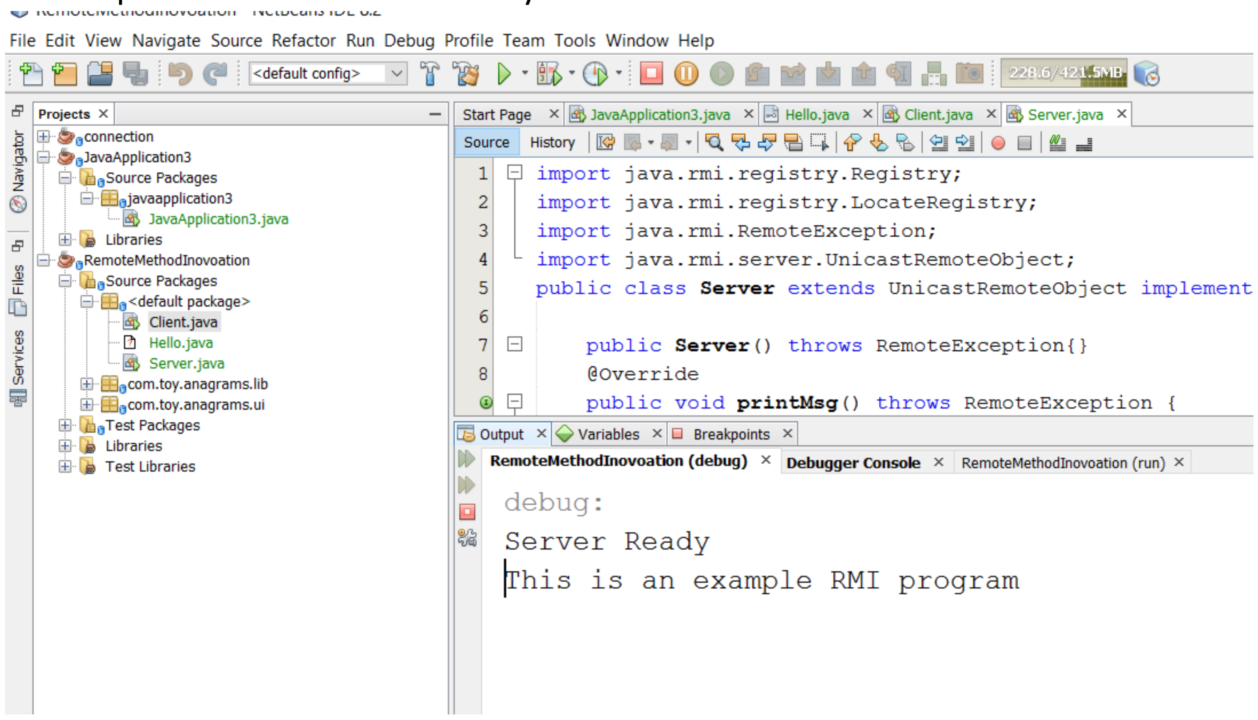
3) ***Client-code***:

```
 import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
import java.util.Scanner;
public class Client {
   private Client() {}
   public static void main(String[] args) throws RemoteException,NotBoundException {
      Client c = new Client();
      c.connectRemote();
   }
   private void connectRemote() throws RemoteException, NotBoundException {
      try {
         Registry registry = LocateRegistry.getRegistry("Localhost",8000);
         Hello h = (Hello)registry.lookup("Hello server");
         System.out.println("In client");
         h.printMsg();
         Scanner sc = new Scanner(System.in);
         System.out.println("Enter two integer values");
         int a = sc.nextInt();
         int b = sc.nextInt();
         System.out.println("Sum is: "+h.adder(a,b));
      }
      catch(RemoteException e){
         System.out.println("Exception: "+e);
      }   } }
```

Output:

1) First, we need to compile all the files.

2) Then run the server-code, After starting the server you can see the message in the output screen as “server Ready”..

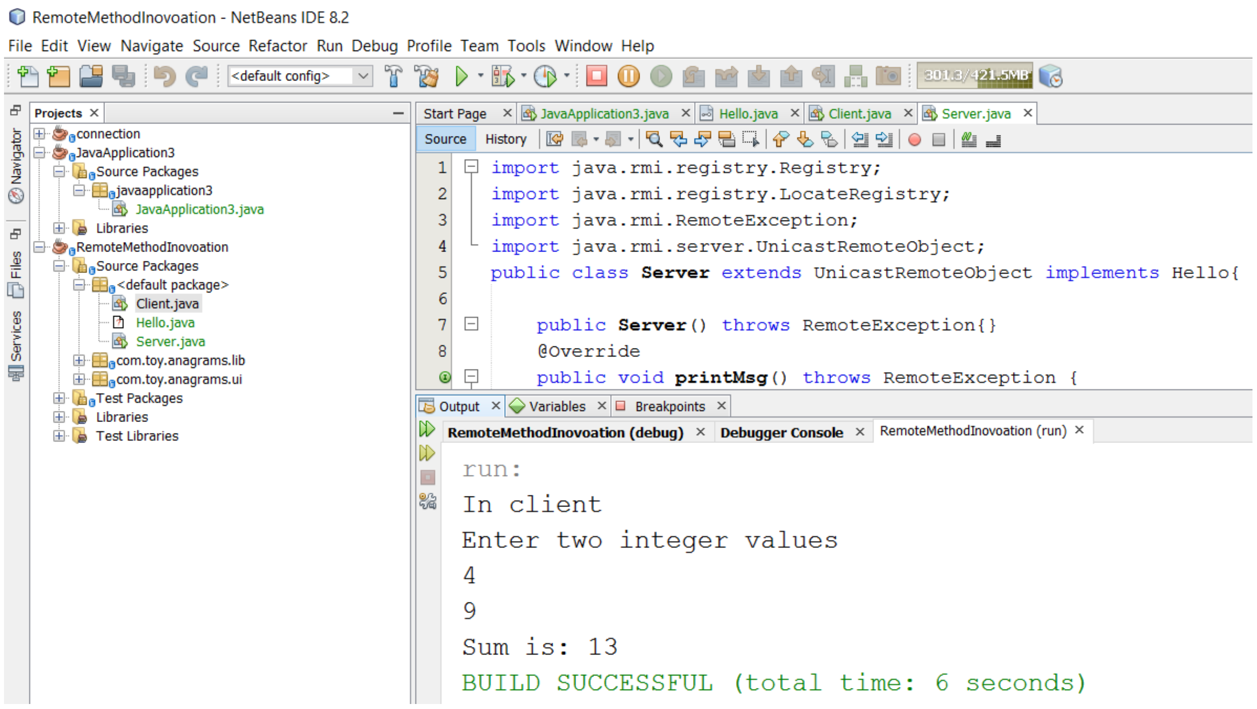


```java
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Server extends UnicastRemoteObject implement

    public Server() throws RemoteException{}
    @Override
    public void printMsg() throws RemoteException {
```

```
debug:
Server Ready
This is an example RMI program
```

3) Now, run the client-code. And you can enter the values of the variables so, that it will add both the values.



```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Server extends UnicastRemoteObject implements Hello{

    public Server() throws RemoteException{}
    @Override
    public void printMsg() throws RemoteException {
```

```
run:
In client
Enter two integer values
4
9
Sum is: 13
BUILD SUCCESSFUL (total time: 6 seconds)
```

Conclusion: The Remote Method Invocation program is implemented successfully.