**Experiment No 1**

**AIM** : Client server based program using RMI (Remote Method Invocation).

**Theory :**

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. Let's understand the stub and skeleton objects:

**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:

* It initiates a connection with remote Virtual Machine (JVM),
* It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
* It waits for the result
* It reads (unmarshals) the return value or exception, and
* 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:

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

CLIENT SERVER



**STEPS TO RUN RMI PROGRAM :**

1. compile all the java files.
2. create stub and skeleton object by rmic tool.
3. start rmi registry in one command prompt.
4. start the server in another command prompt.
5. start the client application in another command prompt.

**Program :**

1. **Adder File :**

import java.rmi.\*;

public interface Adder extends Remote

{

public int add(int x,int y)throws RemoteException;

}

1. **AdderRemote File** :

import java.rmi.\*;

import java.rmi.server.\*;

public class AdderRemote extends UnicastRemoteObject implements Adder

{

AdderRemote() throws RemoteException

{

super();

}

public int add(int x,int y)

{

return x+y;

}

}

1. **MyClient File** :

import java.rmi.\*;

import java.util.Scanner;

public class MyClient

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter no. to add");

int a = sc.nextInt();

int b = sc.nextInt();

try

{

Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");

System.out.println(stub.add(a,b));

}

catch(Exception e){System.out.println(e);

}

}

}

1. **MyServer File :**

import java.rmi.\*;

import java.rmi.registry.\*;

public class MyServer

{

public static void main(String args[])

{

Try

{

Adder stub=new AdderRemote();

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

}

catch(Exception e)

{

System.out.println(e);

}

}

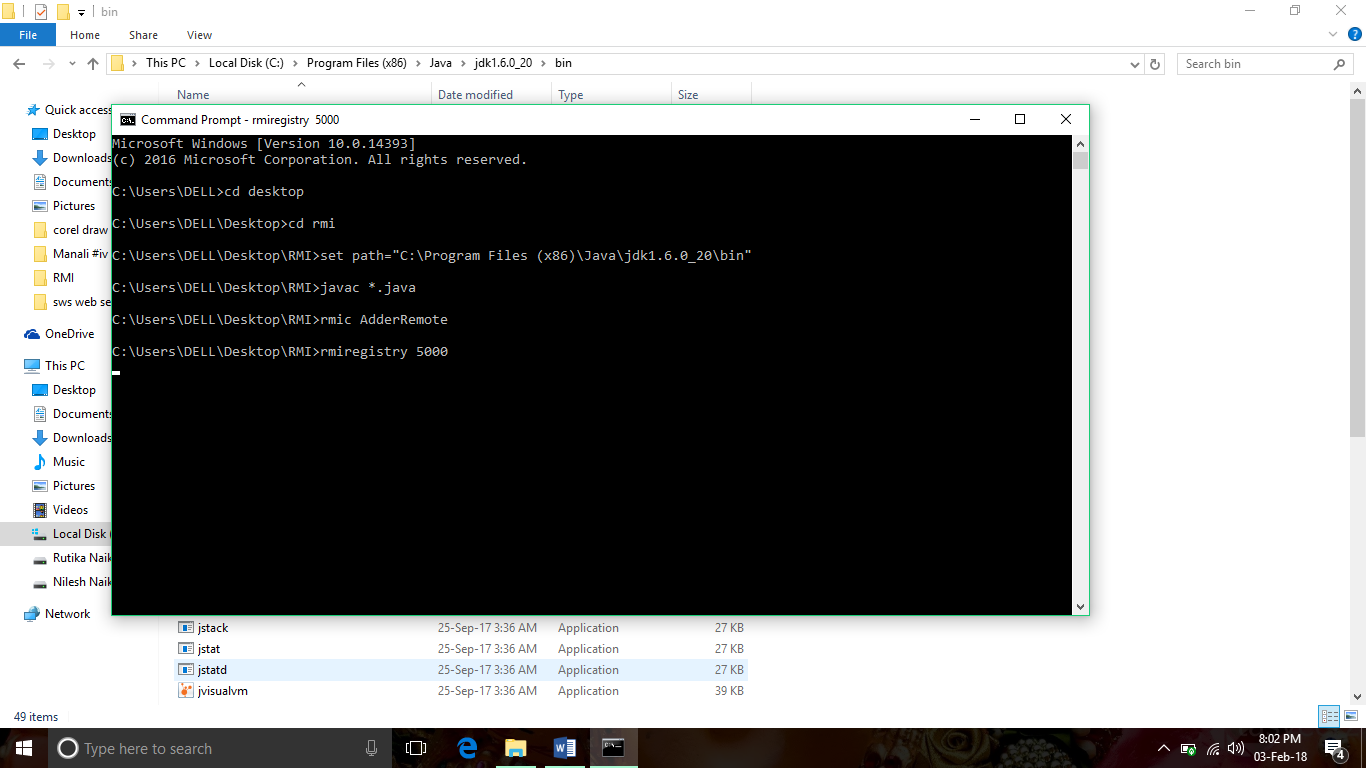
}

**OUTPUT :**

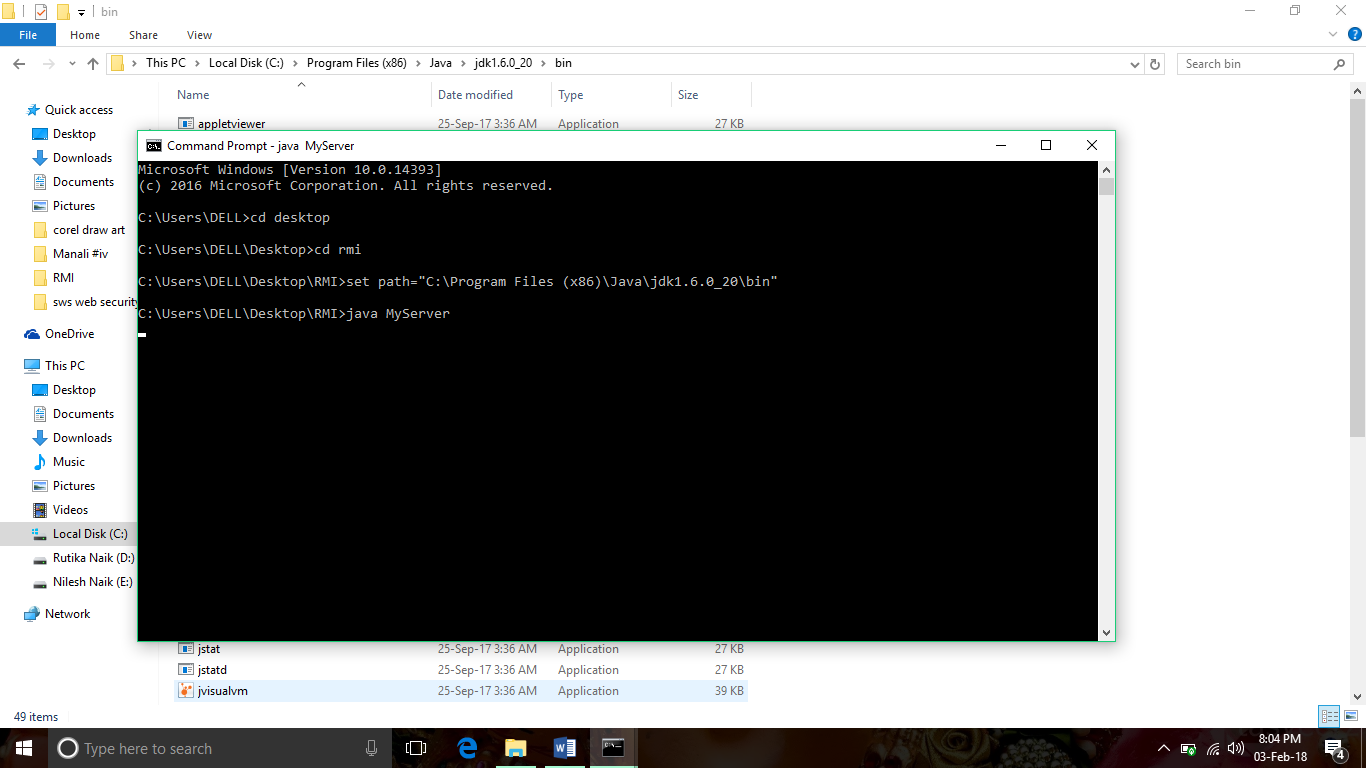
### Compiling all java Files in rmi folder.

### Creating stub and skeleton objects using the rmic tool

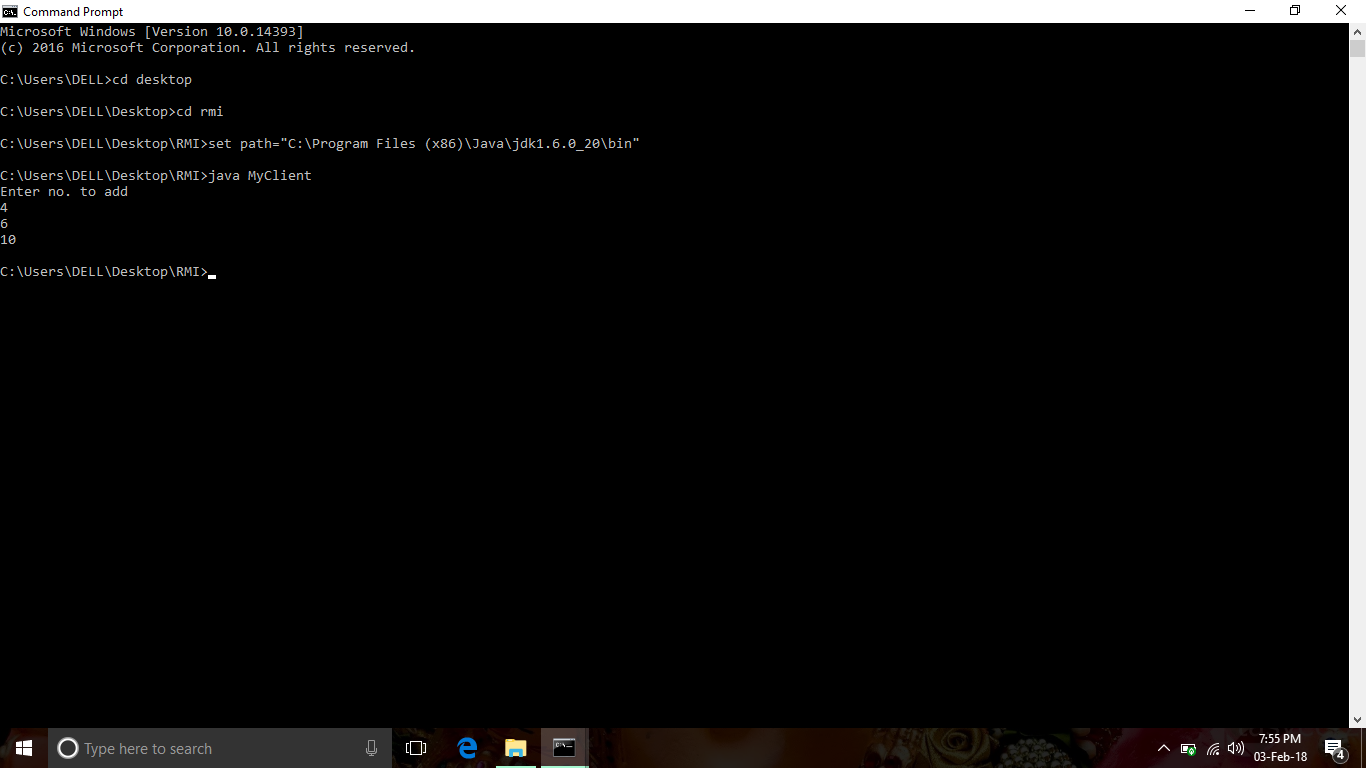
### Start the registry service by the rmiregistry tool.



**Starting Server :**



**Running Client Application :**



**CONCLUSION :**

RMI provides a solid platform for truly object oriented distributed computing.RMI is a very good way to implement distributed systems if they can be build in pure Java.The programmer doesn’t have to care about the distribution during development. So the objects can be distributed dependent on runtime.

Hence , we have studied and implemented RMI successfully.