ANISH ADNANI
D17B / 01

EXPERIMENT 1

Aim: Write a program to design a client server application using Java RMI

Theory

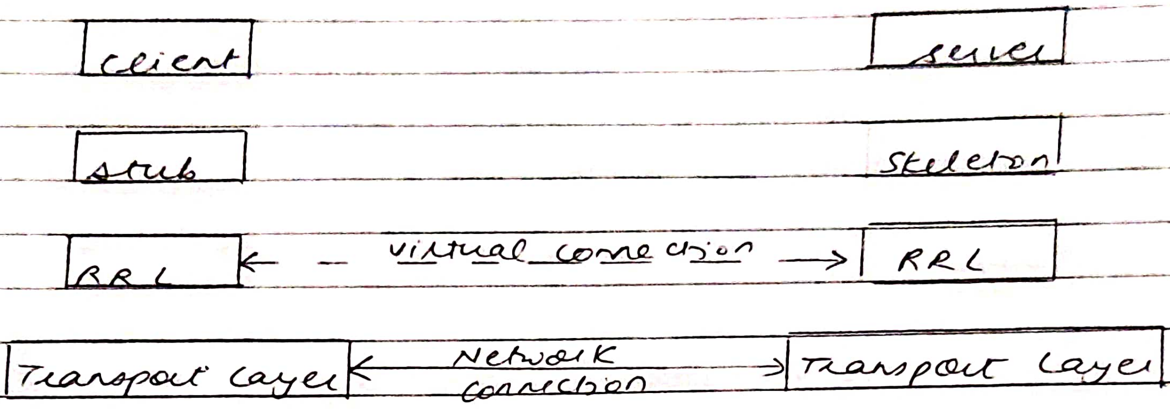
RMI stands for Remote method invocation. It is a mechanism that allows an object residing in one system (JVM) to access / invoke an object running on another JVM

RMI is used to build distributed applications it provides remote communication between Java programs. It is provided in the package java.rmi

- 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.

The following diagram shows the architecture of an RMI application



components of RMI

1) Transport layer: This layer connects the client and server. It manages the existing connection and also sets up new connection

2) stub: A stub is a representation (proxy) of the remote object at client. It resides in client system; it acts as a gateway of the client program

3) skeleton: This is the object which resides on the server side. Stud communicates with this skeleton to pass request to the remote object

4) RRL (Remote Reference layer): It is the layer which manages the reference made by the client to the remote object

- working of an RMI application

The following points summarize how an RMI application works

- When a client makes call to the remote object, it is received by the stub which eventually passes the 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 requests to the RRL on the server side
- The RRL on the server side passes the request to the skeleton (proxy to the server) which finally invokes the required object on the server
- The result is passed all the way back to client

- Marshalling and Unmarshalling

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent out over the network. These parameters maybe of primitive type or object. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are object, then they are serialized. This process is known as marshalling.

At the server side, the packet parameters are unbundled and then the required method is invoked. This process is known as unmarshalling.

steps to write RMI program

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

Commands

```
1) javac *.java
2) rmic AdderRemote
3) rmiregistry 5000
4) java MyServer
5) java MyClient
```

Goals of RMI

Following are the goals of RMI :

1) To minimize the complexity of the application
2) To preserve the safety
3) Distributed garbage collection
4) minimize the difference between working with local and remote object

Conclusion Thus we have successfully implemented and programmed client server application using Java RMI

Creating the remote interface

Calc.java

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
public int sub(int x, int y) throws RemoteException;
public int mul(int x, int y) throws RemoteException;
}
```

Providing implementation of remote interface

CalcRemote.java

```
import java.rmi.*;
import java.rmi.server.*;
public class CalcRemote extends UnicastRemoteObject implements Calc{
CalcRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
public int sub(int x,int y){return x-y;}
public int mul(int x,int y){return x*y;}
}
```

Creating Client file

MyClient.java

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Calc stub=(Calc)Naming.lookup("rmi://localhost:5000/anish");
System.out.println(stub.add(32,4));
}catch(Exception e){}
}
}
```

Creating Server file

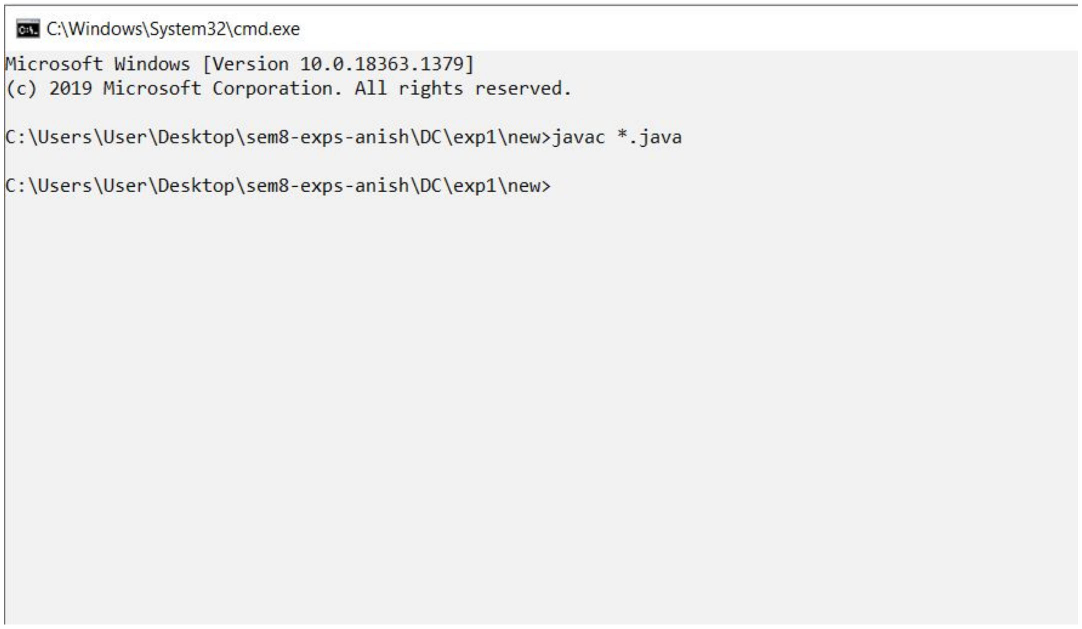
MyServer.java

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
```

```
Calc stub=new CalcRemote();
Naming.rebind("rmi://localhost:5000/anish",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

Implementation Steps

Step1: Compile all java files



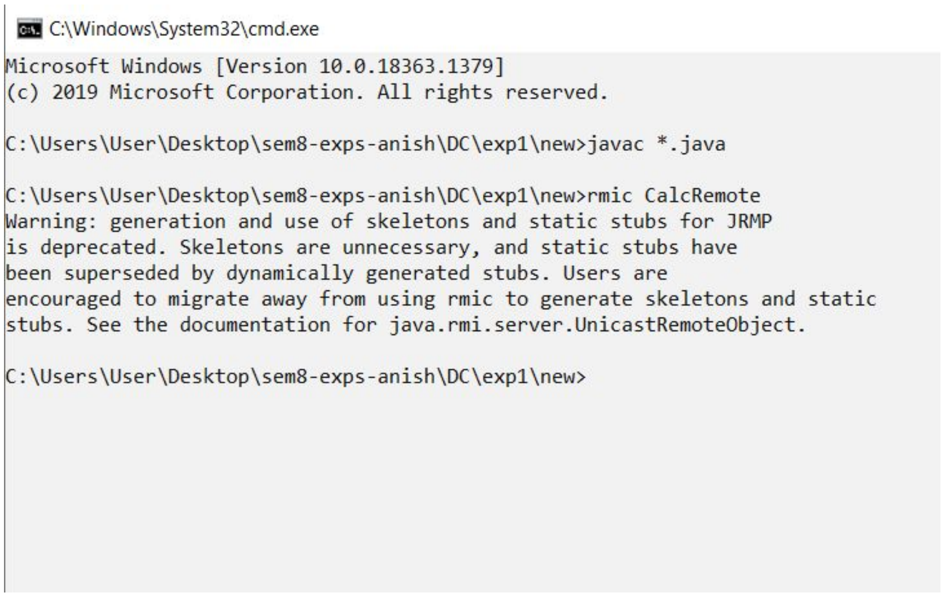
C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>javac *.java

C:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>
```

Step2: create stub and skeleton object by using rmic tool



C:\Windows\System32\cmd.exe

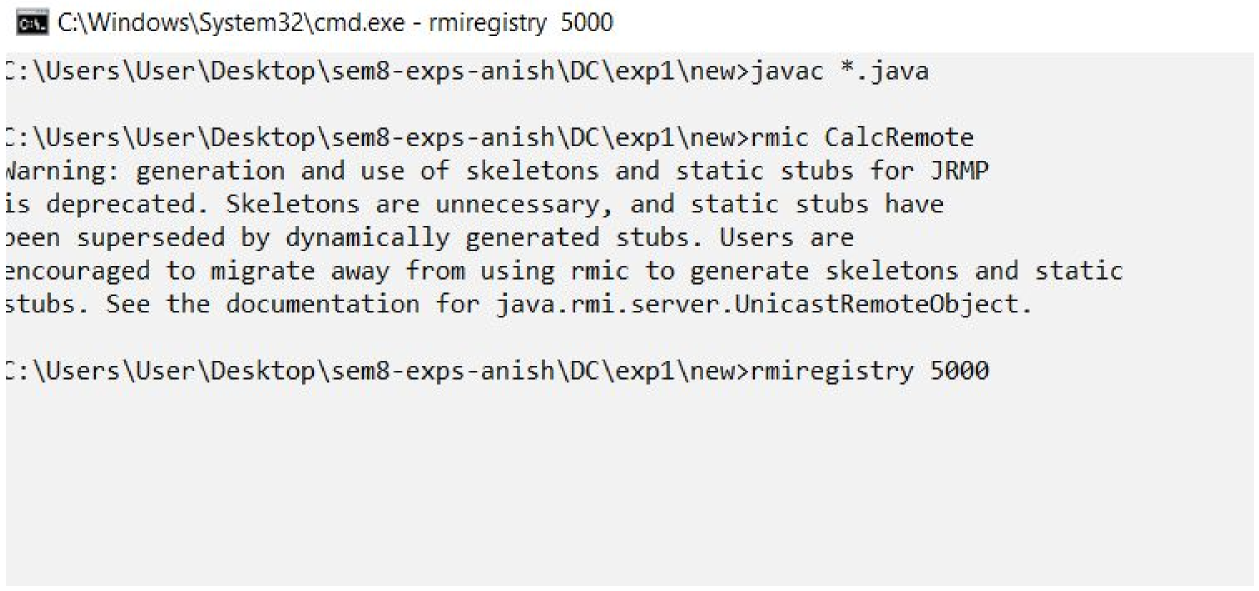
```
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>javac *.java

C:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>rmic CalcRemote
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

C:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>
```

Step3: start rmi registry in one command prompt

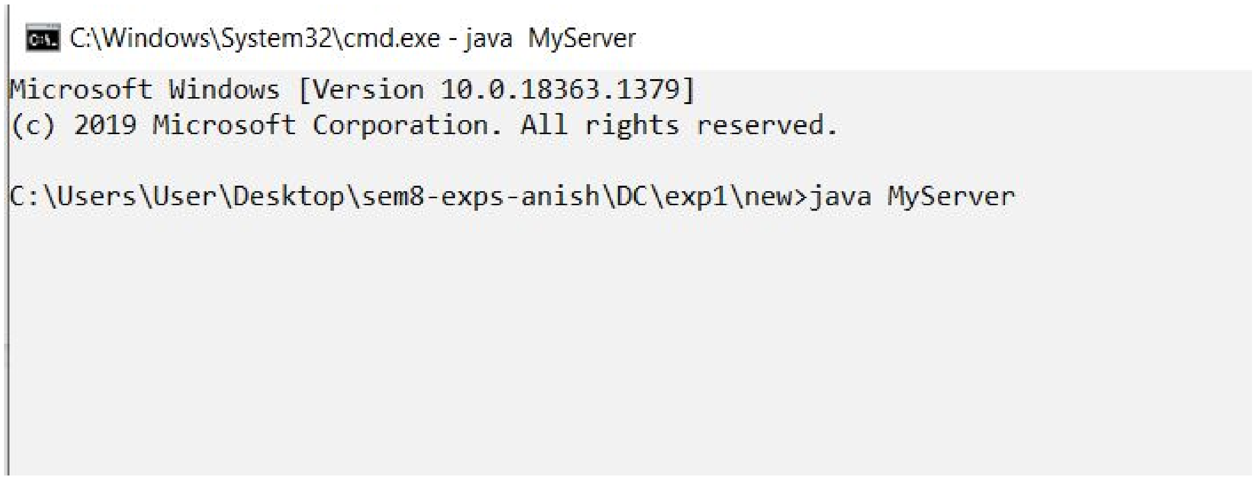


```
C:\Windows\System32\cmd.exe - rmiregistry  5000
:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>javac *.java

:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>rmic CalcRemote
Narning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>rmiregistry 5000
```

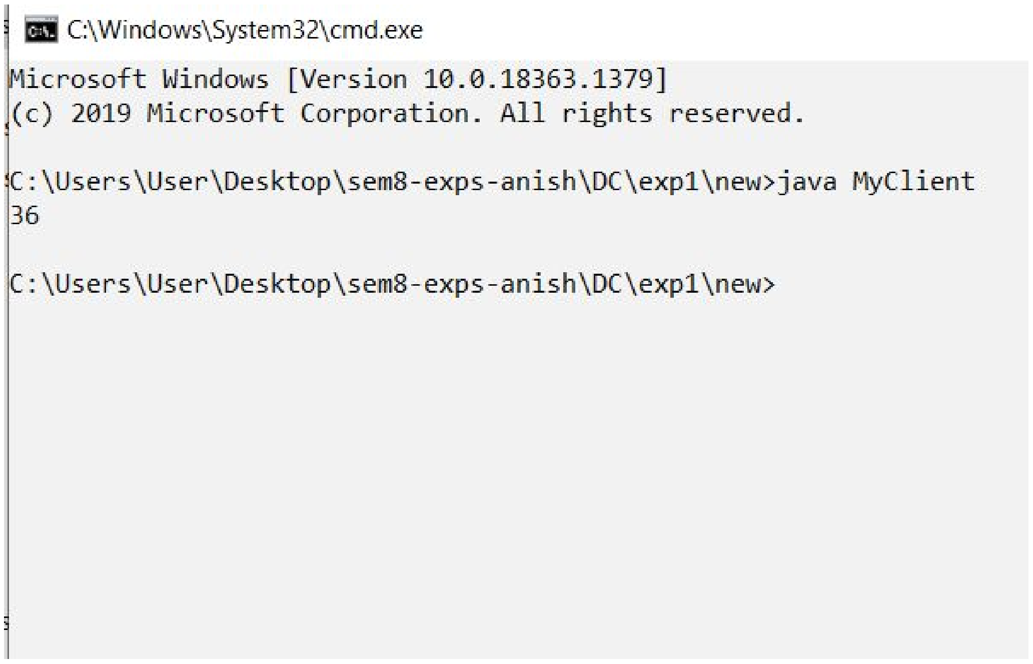
Step4: Start the server in another command prompt



```
C:\Windows\System32\cmd.exe - java  MyServer
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>java MyServer
```

Step5: Start Client in another command prompt


C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>java MyClient
36

C:\Users\User\Desktop\sem8-exps-anish\DC\exp1\new>
```