

Name: Shivam Tiwari

Roll no : 5117060

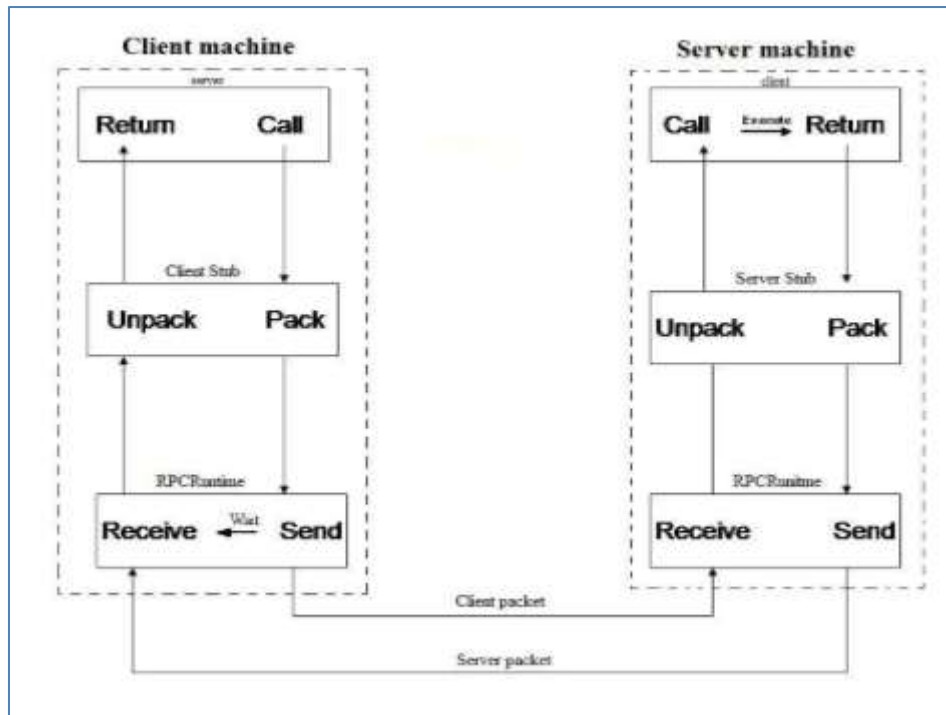
EXPERIMENT 02

Aim: To implement RMI using java

Theory:

- RPC mechanism uses the concepts of stubs to achieve the goal of semantic transparency.
- Stubs provide a local procedure call abstraction by concealing the underlying RPC mechanism.
- A separate stub procedure is associated with both the client and server processes.
- RPC communication package known as RPC Runtime is used on both the sides to hide existence and functionalities of a network.
- Thus, implementation of RPC involves the five elements of program:
 - Client
 - Client Stub
 - RPC Runtime
 - Server stub
 - Server

The client, the client stub, and one instance of RPC Run time execute on the client machine. The server, the server stub, and one instance of RPC Run time execute on the server machine. Remote services are accessed by the user by making ordinary LPC.



1. Client

- A Client is a user process which initiates a RPC
- The client makes a normal call that will invoke a corresponding procedure in the client stub.

2. Client Stub

Client stub is responsible for the following two tasks:

- On receipt of a call request from the client, it packs specifications of the target procedure and arguments into a message and asks the local RPCRuntime to send it to the server stub.
- On receipt of the result of procedure execution, it unpacks the result and passes it to the client.

3. RPCRuntime

- Transmission of messages between Client and the server machine across the network is handled by RPCRuntime.
- It performs Retransmission, Acknowledgement, Routing and Encryption.

- RPCRuntime on Client machine receives messages containing result of procedure execution from server and sends it client stub as well as the RPCRuntime on server machine receives the same message from server stub and passes it to client machine.
- It also receives call request messages from client machine and sends it to server stub.

4. Server Stub

Server stub is similar to client stub and is responsible for the following two tasks:

- i. On receipt of a call request message from the local RPCRuntime, it unpacks and makes a normal call to invoke the required procedure in the server.
- ii. On receipt of the result of procedure execution from the server, it unpacks the result into a message and then asks the local RPCRuntime to send it to the client stub.

5. Server

When a call request is received from the server stub, the server executes the required procedure and returns the result to the server stub.

RMI:

Server Side:

- Create a remote interface. (`myInterface.java`)
- Create a separate java file to implement the remote interface. (`RMIServer.java`)
- Register the interface in RMI registry (bind with any name eg: `myRMIService`)

Client Side:

- In client code, through interface name (in our case `myRMIService`) create a "fake remote" object reference of server within client. (`RMIClient.java`)
- Through this reference access the server methods / services

CODE:

. MyInterface.java

```
1. import java.rmi.*;
2. public interface MyInterface extends Remote
3. {
4.     public String countInput(String input)throws
        RemoteException;
5. }
```

2. RMIServer.java

```
1. import java.rmi.*;
2. import java.rmi.server.*;
3. public class RMIServer extends UnicastRemoteObject
    implements MyInterface
4. {
5.     public RMIServer()throws RemoteException
6.     {
7.         System.out.println("Remote Server is running
            Now.!!");
8.     }
9.     public static void main(String arg[])
10. {
11.     try{
12.         RMIServer p=new RMIServer();
13.         Naming.rebind("rmiInterface",p);
14.     }
15. catch(Exception e)
16. { System.out.println("Exception occurred :
        "+e.getMessage()); }
17. }
```

```

18.
19.     @Override
20.     public String countInput(String input) throws
        RemoteException
21.     {
22.         System.out.println("Received your input "+ input+" at
            server!!");
23.         String reply;
24.         reply="You have typed "+ input.length() +"
            letters!!";
25.         return reply;
26.     }
27. }

```

3. RMIClient.java

```

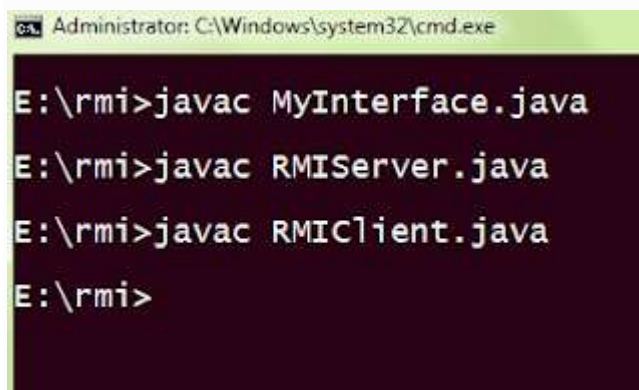
1. import java.rmi.*;
2. import java.io.*;
3. public class RMIClient
4. {
5.     public static void main(String args[])
6.     {
7.         try
8.         { BufferedReader br=new BufferedReader(new
            InputStreamReader(System.in));
9.             MyInterface p=(
            MyInterface)Naming.lookup("rmiInterface");
10.             System.out.println("Type something...");
11.             String input=br.readLine();
12.             System.out.println(p.countInput(input));
13.         }
14.         catch(Exception e) {

```

```
15.         System.out.println("Exception occurred :  
        "+e.getMessage() );  
16.     }  
17. }  
18. }
```

RESULT:

Compile the java files.



```
Administrator: C:\Windows\system32\cmd.exe  
E:\rmi>javac MyInterface.java  
E:\rmi>javac RMIServer.java  
E:\rmi>javac RMIClient.java  
E:\rmi>
```

Run remote object registry called `rmiregistry`. It is used by RMI servers on the same host to bind remote objects to names.



```
Administrator: C:\Windows\system32\cmd.exe - rmiregistry  
E:\rmi>rmiregistry
```

Run the Server class.



```
Administrator: C:\Windows\system32\cmd.exe - java RMIServer  
E:\rmi>java RMIServer  
Remote Server is running Now.!!
```

Run the client class. You can see it is prompting for input



```
Administrator: C:\Windows\system32\cmd.exe - java RMIClient  
E:\rmi>java RMIClient  
Type something...
```

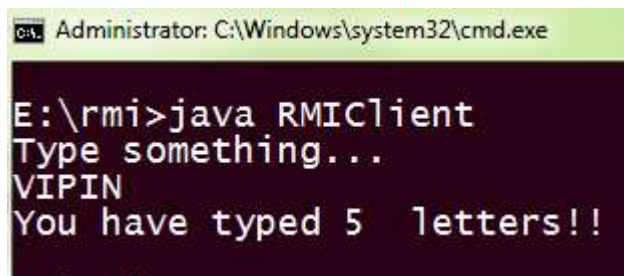
Type some text

Input is received by remote server



```
Administrator: C:\Windows\system32\cmd.exe - java RMIServer  
E:\rmi>java RMIServer  
Remote Server is running Now.!!  
Received your input VIPIN at server!!
```

Client receives reply from remote server



```
Administrator: C:\Windows\system32\cmd.exe  
E:\rmi>java RMIClient  
Type something...  
VIPIN  
You have typed 5 letters!!
```