**TRIBHUVAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

**ADVANCED COLLEGE OF ENGINEERING AND MANAGEMENT**

**KUPONDOLE, LALITPUR**



# LAB REPORT

LAB NO : 2

SUBJECT: Distributed System

**SUBMITTED BY: SAMEEP DHAKAL**

**SUBMITTED TO: DEPARTMENT OF COMPUTER ENGINEERING** DATE:2021-06-20

**TITLE: Remote Method Invocation in JAVA**

**1.Objective:**

The objective of this lab is to learn about RMI and implement a simple client side and server-side system using java.

**2.Software Used:**

The Java Development Kit (JDK) was used as a compiler, notepad was used to write the code and the windows command prompt was used to execute the code.

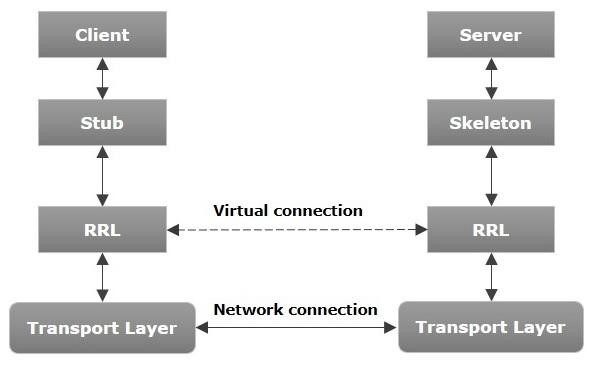
**3.Introduction**

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.

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.



* 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.

**4.Code Implementation**

Client side code :

import java.rmi.Naming; public class RmiClient

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

RmiServerIntf obj = (RmiServerIntf)Naming.lookup("//localhost/RmiServer");

//returns a reference, a stub, for the remote object associated with the specified name, i.e, RmiServer in //this case.

System.out.println(obj.getMessage());

}

}

Server side code :

import java.rmi.Naming; import java.rmi.RemoteException; import java.rmi.registry.LocateRegistry; import java.rmi.server.UnicastRemoteObject;

public class RmiServer extends UnicastRemoteObject implements RmiServerIntf { public static final String MESSAGE = "Hello World";

public RmiServer() throws RemoteException { super(0); // required to avoid the 'rmic' step, see below

}

public String getMessage() { return MESSAGE;

}

public static void main(String args[]) throws Exception { System.out.println("RMI server started"); try {

//special exception handler for registry creation

LocateRegistry.createRegistry(1099);

System.out.println("java RMI registry created.");

}

catch (RemoteException e) {

// do nothing, error means registry already exists

System.out.println("java RMI registry already exists.");

}

//Instantiate RmiServer

RmiServer obj = new RmiServer();

// Bind this object instance to the name "RmiServer"

Naming.rebind("//localhost/RmiServer", obj);

System.out.println("PeerServer bound in registry"); }

}

Server Interface side code :

import java.rmi.Remote;

import java.rmi.RemoteException;

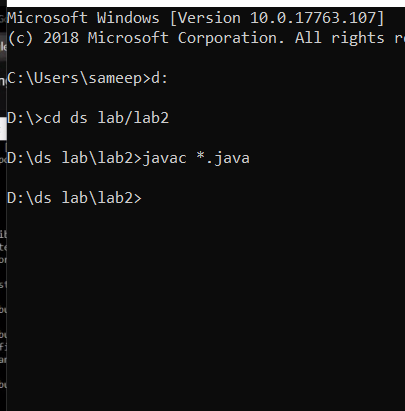
public interface RmiServerIntf extends Remote { public String getMessage() throws RemoteException;

}

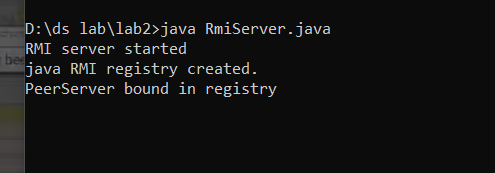
**5.Result**

The following results were observed when the code was executed.

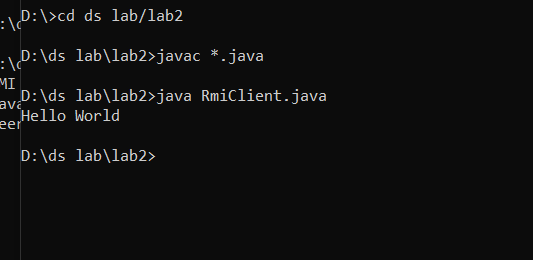
First the rmi server and client code,interfae code was complied.



Server code was executed



Client code was executed



Operation on the client was observed as Hello World.

**6.Discussion**

We implemented a simple RMI using java and observed the following limitations:

* It is hard to tell which objects are local and which are remote.
* Less efficient than socket objects.
* Assuming the default threading will allow ignoring the coding, being the servers are thread- safe and robust.
* It cannot use the code out of the scope of java.
* Security issues need to be monitored more closely.

**7.Conclusion**

RMI provides a solid platform for truly object-oriented distributed computing. Hence, in this lab we have used RMI to implement a simple client side and server-side system using java.