Madan Bhandari Memorial College

Department of Computer Science and Information Technology (B.Sc.CSIT)

Ninayak Nagar, New Baneshwor, Kathmandu

Practical Sheet

Submitted By:- Utsav Acharya Sharma

Submitted To Debesh Adhikari

Submission Date:- 2080/09/20

Program No:- 13

Lab Date:- 2080/09/19

T.U.Roll.No. :- 24179

Title: Remote Method Invocation (RMI)

Introduction:

RMI:

RMI stands for Remote Method Invocation, and it is a Java-based technology that allows the execution of methods on remote objects as if they were local.

RMI Architecture:

i) Stub : The client-side proxy for the remote object.

ii) Skelchn: The server-side proxy works as a gateway for incoming requests.

iii) Remote Reference Layer (RRL): Provides support for multiple communication protocals.

Marshalling:

The process of converting the data into a format that can be transmitted over the network. In RMI, packaging the method parameter into byte stream.

Unmarshalling

The process of reconstructing the data on the receiving end. In RMI, converting byte stream back into method parameters.

RMI Registry:

The RMI Registry is a simple naming service that binds remote objects to names, making them accessible to clients. It acts as a central repository for remote object references.

Steps to create an RMI Application

i) Define the Remote Interface
ii) Implement the Remote Interface
iii) Compile the Interface and Implementation
iv) Generate Stub and Skeleton
v) Start RMI Registry
vi) Run Server
vii) Run Client.

Difference between RMI and CORBA

i) Language Specificity: RMI is Java specific while CORBA is a platform independent.

ii) Object Activation: RMI uses lightweight approach to object activation, CORBA has more complex approach.

iii) Interface Definition: RMI uses Java interfaces for defining remote interfaces, while CORBA uses the IDL.

iv) Object References: RMI uses a simple object reference mechanism, whereas CORBA uses more complex object references.

Task I Code:

addI.java

```
import java.rmi.Remote;
impat java.rmi.*;
public interface addI extends Remote{
    public int add(int x, int y) throws Exception;
}
```

AddC.Java

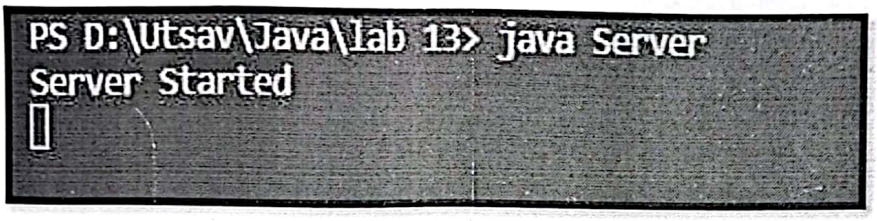
```
impat java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class AddC extends UnicastRemoteObject implements addI{
    public AddC() throws Exception{
        super();
    }
    public int add(int x,int y){
        return x+y;
    }
}
```

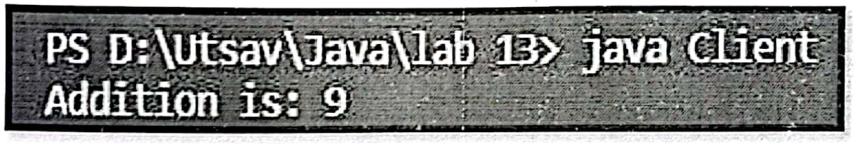
Client.Java

```
import java.rmi.*;
public class Client{
    public static void main(String a[]) throws Exception{
        addI obj = (addI) Naming.lookup("ADD");
        int n= obj.add(5,4);
        System.out.println("Addition is: "+n);
    }
}
```

Server.Java

```
import java.rmi.*;
public class Server{
    public static void main (String a[]) throws Exception{
        AddC obj = new AddC();
        Naming.rebind ("ADD", obj);
        System.out.println ("Server Started");
    }
}
```


PS D:\Utsav\Java\lab 13> java Server
Server Started


PS D:\Utsav\Java\lab 13> java Client
Addition is: 9

Task II Code:

mathI.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface mathI extends Remote {
        int add(int a, int b) throws RemoteException;
        int subtract (int a, int b) throws RemoteException;
        int multiply (int a, int b) throws RemoteException;
        double divide (double a, double b) throws RemoteException;
        int factorial (int n) throws RemoteException;
}
```

MathC.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class MathC extends UnicastRemoteObject implements mathI {
        protected MathC() throws RemoteException {
                super();
        }
        @Override
          public int add (int a, int b) throws RemoteException {
                  return a+b;
          }
        @Override
          public int subtract(int a, int b) throws RemoteException {
                  return a-b;
          }
         @Override
          public int multiply (int a, int b) throws RemoteException {
                  return a*b;
          }
         @Override
          public double divide(double a, double b) throws RemoteException {
           if(b!=0) { return a/b; }
           else { throw new RemoteException ("Cannot divide by 0"); }
         }
```

```
    @Override
    public int factorial(int n) throws RemoteException{
        if(n==0 || n==1){
            return 1;
        }else{
            return n * factorial(n-1);
        }
    }
}
```

Server.Java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Server{
    public static void main(String[] args){
        try{
            Registry registry = LocateRegistry.createRegistry(8086);
            MathC mathObj = new MathC();
            registry.rebind("Math", mathObj);
            System.out.println("Server is running....");
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Client.java

```
import java.rmi.Naming;
public class Client{
    public static void main(String[] args){
        try{
            MathI mathObj = (MathI) Naming.lookup("//localhost:8086/
                                                    Math");

            System.out.println("Addition: "+ mathObj.add(3,3));
            System.out.println("Subtraction: "+ mathObj.subtract(10,4));
            System.out.println("Multiplication: "+mathObj.multiply(3,6));
            System.out.println("Division: "+mathObj.divide(20.0,4.0));
            System.out.println("Factorial: "+mathObj.factorial(5));
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

```
PS D:\Utsav\Java\lab 13> start rmiregistry
PS D:\Utsav\Java\lab 13> java Server
Server is running...
```

```
PS D:\Utsav\Java\lab 13> java Client
Addition: 8
Subtraction: 6
Multiplication: 30
Division: 5.0
Factorial: 120
```