TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Lab Report

On

WAP TO IMPLEMENT JAVA RMI MECHANISM.

Lab No. 2

Submitted by:

Bikrant Bidari

THA076BCT013

Submitted to:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

THA076BCT013

June, 2023

TITLE: WAP TO IMPLEMENT JAVA RMI MECHANISM.

1. THEORY

RMI (Remote Method Invocation) in Java is a mechanism that allows objects residing in different Java virtual machines (JVMs) to invoke methods on each other remotely. RMI enables distributed computing and facilitates communication between client and server applications.

The key components of RMI are:

Remote Interface: It defines the methods that can be invoked remotely. It extends the java.rmi.Remote interface and declares the remote methods. Each method in the remote interface must throw java.rmi.RemoteException.

Remote Object: It is the implementation of the remote interface. The remote object is responsible for executing the methods invoked remotely. The remote object must extend java.rmi.server.UnicastRemoteObject or use a custom subclass to enable remote method invocation.

Stub: The stub acts as a client-side proxy for the remote object. It resides in the client JVM and communicates with the remote object on the server side. The stub marshals the method invocation requests, sends them to the server, and unmarshals the results.

Skeleton: The skeleton resides on the server side and acts as a server-side proxy for the remote object. It receives the method invocation requests from the client stub, dispatches them to the appropriate remote object, and marshals the results back to the client.

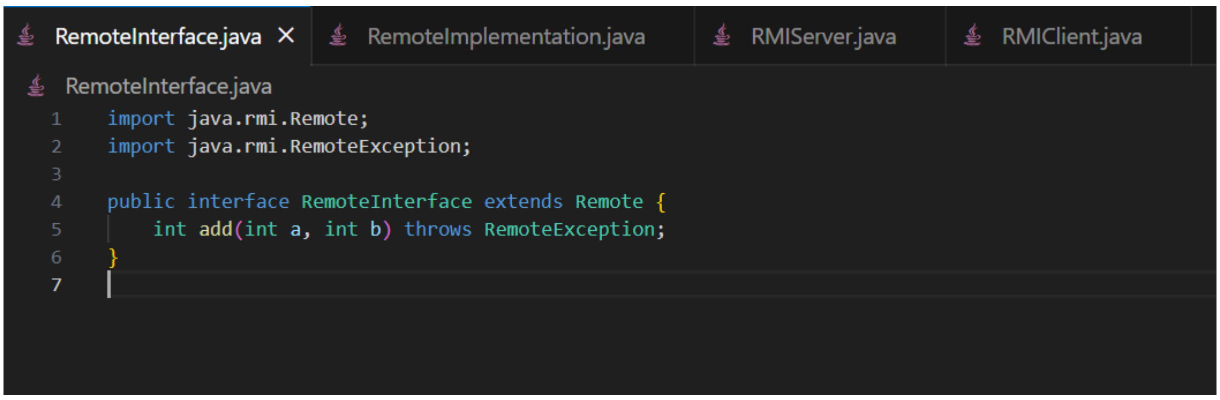
2. PROCEDURE

The steps involved in using RMI include:

a. Design and define the remote interface that declares the methods to be invoked remotely.
b. Implement the remote object that implements the remote interface.
c. Start the RMI registry on the server-side to bind the remote object.
d. Create the client application that looks up the remote object from the RMI registry and invokes its methods remotely.

3. CODE

RemoteInterface.java:

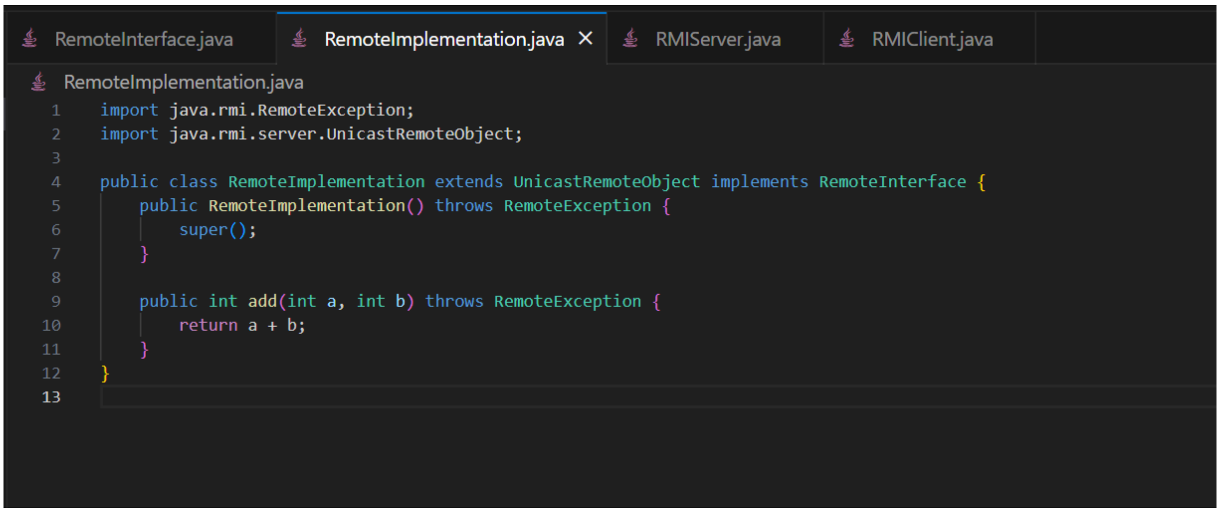


RemoteInterface.java | RemoteImplementation.java | RMIServer.java | RMIClient.java

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

public interface RemoteInterface extends Remote {
    int add(int a, int b) throws RemoteException;
}
```

RemoteImplementation.java:



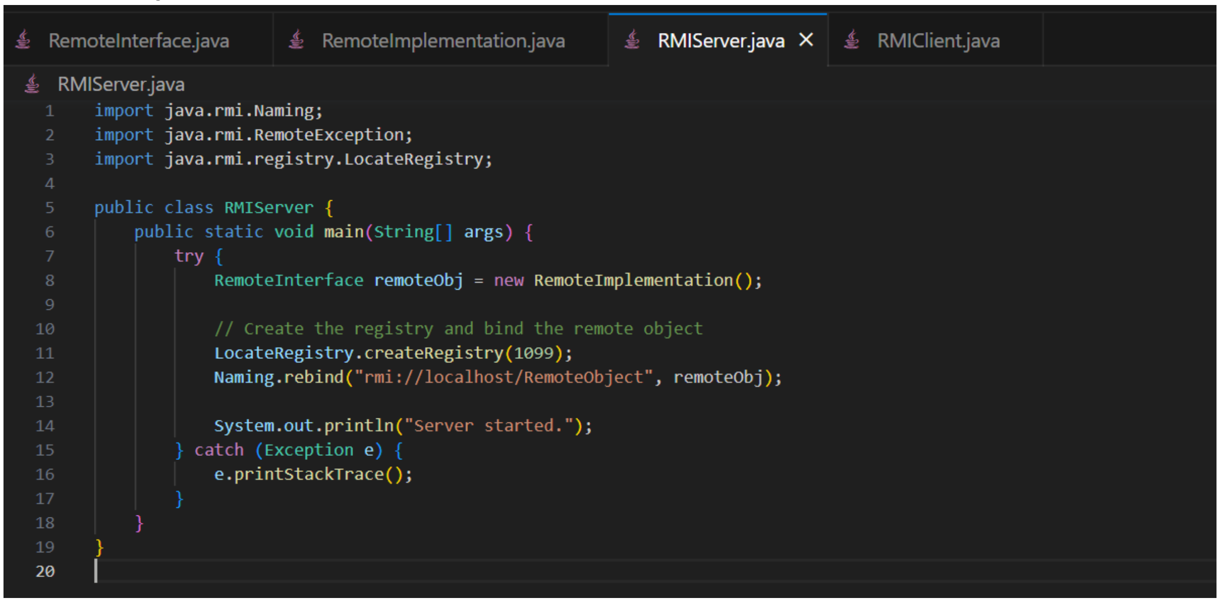
RemoteInterface.java | RemoteImplementation.java | RMIServer.java | RMIClient.java

```java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class RemoteImplementation extends UnicastRemoteObject implements RemoteInterface {
    public RemoteImplementation() throws RemoteException {
        super();
    }

    public int add(int a, int b) throws RemoteException {
        return a + b;
    }
}
```

RMIserver.java:



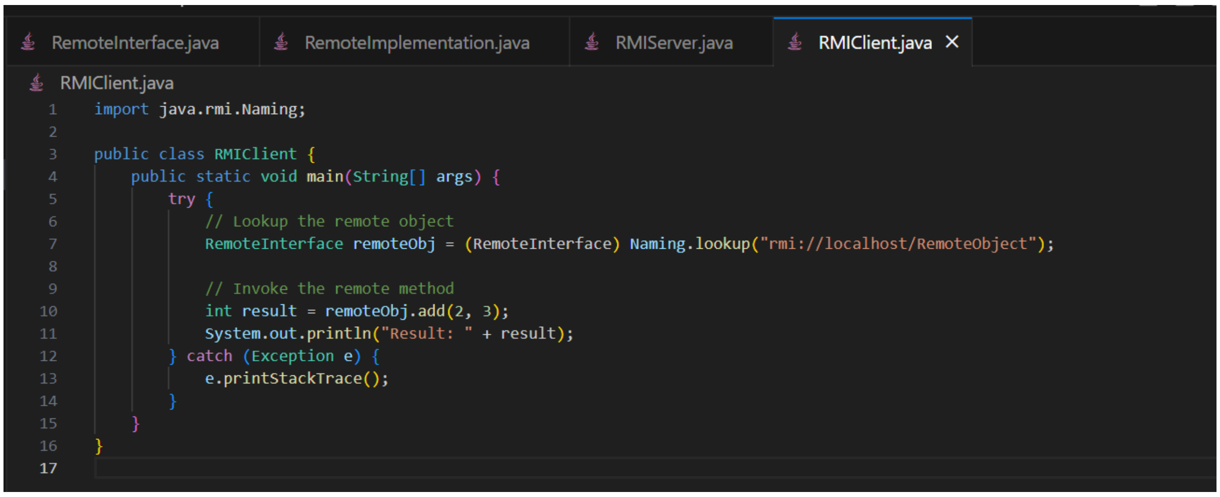
```java
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;

public class RMIServer {
    public static void main(String[] args) {
        try {
            RemoteInterface remoteObj = new RemoteImplementation();

            // Create the registry and bind the remote object
            LocateRegistry.createRegistry(1099);
            Naming.rebind("rmi://localhost/RemoteObject", remoteObj);

            System.out.println("Server started.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

RMIClient.java



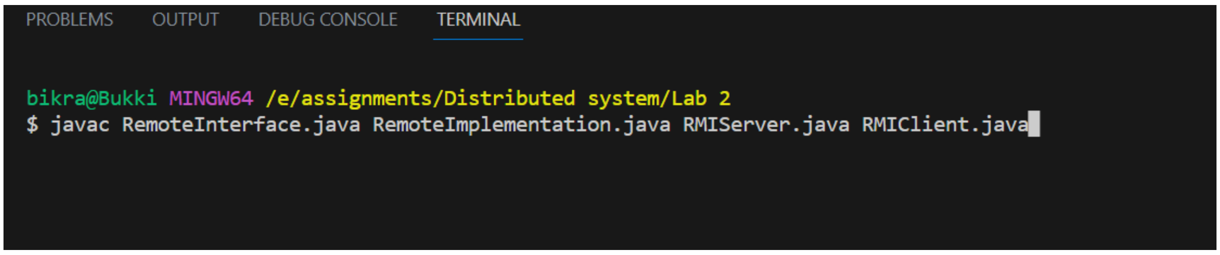
```java
import java.rmi.Naming;

public class RMIClient {
    public static void main(String[] args) {
        try {
            // Lookup the remote object
            RemoteInterface remoteObj = (RemoteInterface) Naming.lookup("rmi://localhost/RemoteObject");

            // Invoke the remote method
            int result = remoteObj.add(2, 3);
            System.out.println("Result: " + result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

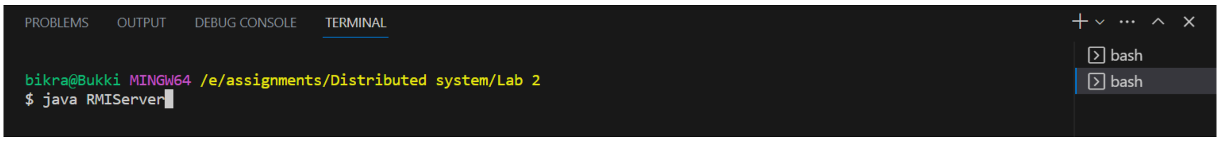
4. OUTPUT

Compile all the Java files using the following command:



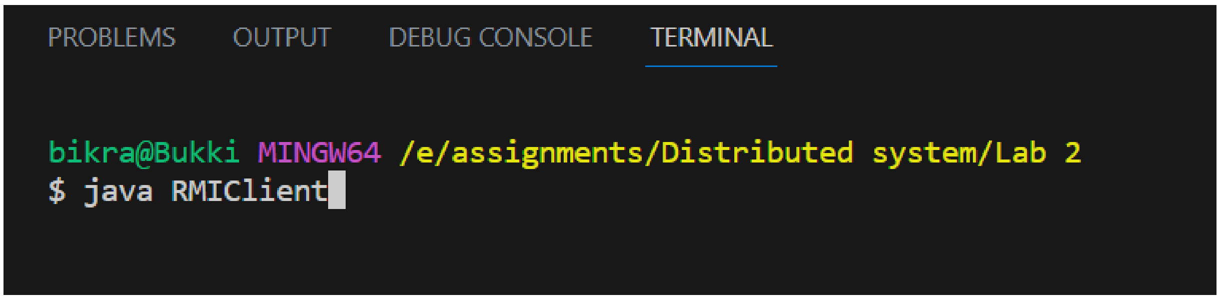
```
bikra@Bukki MINGW64 /e/assignments/Distributed system/Lab 2
$ javac RemoteInterface.java RemoteImplementation.java RMIServer.java RMIClient.java
```

Start the RMI server by running the following command in a separate terminal or command prompt:



```
bikra@Bukki MINGW64 /e/assignments/Distributed system/Lab 2
$ java RMIServer
```

Finally, run the RMI client by executing the following command in another terminal or command prompt:



```
bikra@Bukki MINGW64 /e/assignments/Distributed system/Lab 2
$ java RMIClient
```

Which should give you the output:

“Result:5”

5. CONCLUSION AND DISCUSSION

In this Lab we learnt about RMI and implemented it on JAVA for addition of two numbers.