Lab Report-03

Gandaki College of Engineering and Science

Distributed System

Lab Experiment: Java RMI Implementation

Submitted By:

Name: Dipendra Raut Kurmi

Roll.No: 18

Batch: 2021SE

Submitted To:

Er. Amrit Poudel

Lecturer at Gandaki College Of

Engineering and Science

Objective:

To implement remote method invocation between a client and server using ava RMI.

Theory:

Java RMI (Remote Method Invocation) allows an object residing in one Java Virtual Machine (JVM) to invoke methods on an object located in another JVM. It provides a simple and effective way for developing distributed applications in Java.

Features of Java RMI:

- Provides distributed object communication.
- Supports remote object invocation.
- Allows passing of complex objects between JVMs.
- Built-in Java security manager.
- Uses Java serialization.

Code:

// RMI Interface (Hello.java)

```
import java.rmi.Remote;

import java.rmi.RemoteException;

public interface Hello extends Remote {
String sayHello() throws RemoteException;
}
```

// RMI Server Implementation (HelloImpl.java)

```
import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {
protected HelloImpl() throws RemoteException {
super();
}

public String sayHello() throws RemoteException {
return "Hello, this is a remote method call!";
}
}
```

// Server Program (Server.java)

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

import java.rmi.registry.Registry;

public class Server {
public static void main(String[] args) {
try {
HelloImpl obj = new HelloImpl();
Registry registry = LocateRegistry.getRegistry();
registry.rebind("Hello", obj);

System.out.println("Server is ready...");
} catch (Exception e) {
System.err.println("Server exception: " + e.toString());
e.printStackTrace();
}
}
}
```

// Client Program (Client.java)

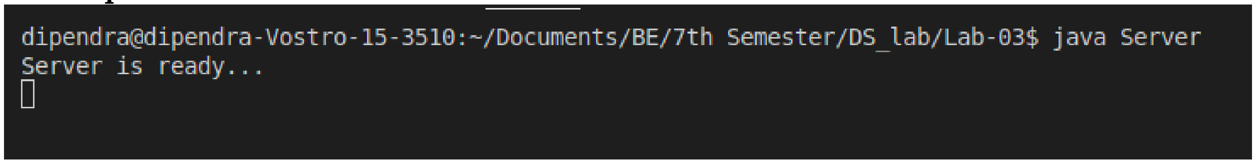
```java
import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

public class Client {
public static void main(String[] args) {
try {
Registry registry = LocateRegistry.getRegistry("localhost", 1099);
Hello stub = (Hello) registry.lookup("Hello");
String response = stub.sayHello();
System.out.println("Response from server: " + response);
} catch (Exception e) {
System.err.println("Client exception: " + e.toString());
e.printStackTrace();
}
}
}
```

Result:

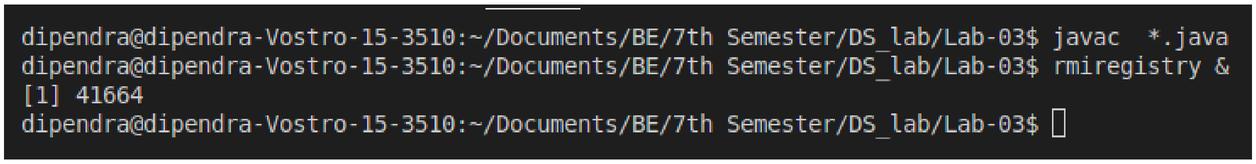
When running the Server.java and Client.java programs:

1. The server starts and binds the remote object.
2. The client looks up the remote object and invokes the sayHello() method.
3. The server returns a greeting message.
4. The client receives and displays the response.

Server Output:

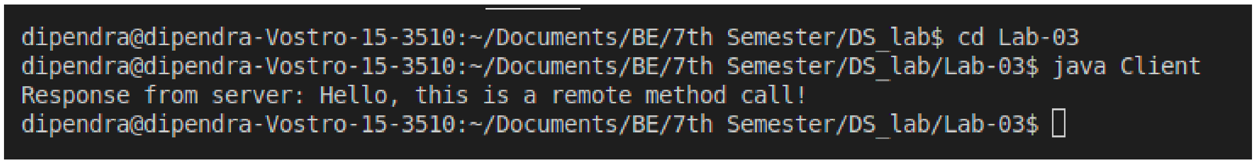


```
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-03$ java Server
Server is ready...
```



```
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-03$ javac  *.java
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-03$ rmiregistry &
[1] 41664
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-03$ 
```

Client Output:



```
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab$ cd Lab-03
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-03$ java Client
Response from server: Hello, this is a remote method call!
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-03$ 
```

Conclusion:

Hence, we successfully implemented remote method invocation between a client and server using Java RMI. This lab helped us understand distributed object communication and how RMI enables simple and secure interaction between Java programs across different JVMs.