

**Vidyavardhini’s College of Engineering & Technology**

Department of Computer Engineering Academic Year : 2024-25

|  |  |  |  |
| --- | --- | --- | --- |
| **Class:** | **BE** | **Semester:** | **VIII** |
| **Course Code:** | **CSL801** | **Course Name:** | **Distributed Computing Lab** |

|  |  |
| --- | --- |
| **Name of Student:** | **Pratima Dinkar Bombe** |
| **Roll No. :** | **07** |
| **Division:** | **-** |
| **Experiment No.:** | **02** |
| **Title of Experiment:** | **Client/Server using RPC/RMI** |
| **Date of Submission:** | **20/01/2025** |
| **Date of Correction:** | **28/01/2025** |

Evaluation

|  |  |  |
| --- | --- | --- |
| **Performance Indicator** | **Max. Marks** | **Marks Obtained** |
| Performance | 5 |  |
| Understanding | 5 |  |
| Journal work and timely submission | 10 |  |
| Total | 20 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Performance Indicator** | **Exceed Expectations (EE)** | **Meet Expectations (ME)** | **Below Expectations (BE)** |
| Performance | 4-5 | 2-3 | 1 |
| Understanding | 4-5 | 2-3 | 1 |
| Journal work and timely submission | 8-10 | 5-8 | 1-4 |

**Checked by**

**Name of Faculty : Ms. Swati Varma**

**Signature :**

**Date :**

EXPERIMENT 2

**Aim:** To implement Client/Server using RPC/RMI

**Objective:** Develop a program to implement Client/Server using RPC/RMI

**Theory:**

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. It allows an object to invoke methods on an object running in another JVM. It provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton:

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

Stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

A diagram of a application layer

Description automatically generated

**Steps to write the RMI program**

6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

**Code:**

**SampleServer.java**

import java.rmi.\*;

public interface SampleServer extends Remote {

public int sum(int a, int b) throws RemoteException;

}

**SampleServerImpl.java**

import java.rmi.\*;

import java.rmi.server.\*;

import java.rmi.registry.\*;

public class SampleServerImpl extends UnicastRemoteObject implements SampleServer {

SampleServerImpl() throws RemoteException {

super();

}

public int sum(int a, int b) throws RemoteException {

return (a + b);

}

public static void main(String args[]) {

try {

System.setSecurityManager(new SecurityManager());

// Set the security manager

// Create a local instance of the object

SampleServerImpl server = new SampleServerImpl();

// Bind the local instance in the registry

Naming.rebind("SAMPLE-SERVER", server);

System.out.println("Server waiting.....");

} catch (java.net.MalformedURLException me) {

System.out.println("Malformed URL: " + me.toString());

} catch (RemoteException re) {

System.out.println("Remote exception: " + re.toString());

}

}

}

**SampleClient.java**

import java.rmi.\*;

public class SampleClient {

public static void main(String[] args) {

// Set the security manager for the client

System.setSecurityManager(new SecurityManager());

// Get the remote object from the registry

try {

System.out.println("Security Manager loaded");

String url = "//localhost/SAMPLE-SERVER";

SampleServer remoteObject = (SampleServer) Naming.lookup(url);

System.out.println("Got remote object");

System.out.println("1 + 2 = " + remoteObject.sum(1, 2));

} catch (RemoteException exc) {

System.out.println("Error in lookup: " + exc.toString());

} catch (java.net.MalformedURLException exc) {

System.out.println("Malformed URL: " + exc.toString());

} catch (java.rmi.NotBoundException exc) {

System.out.println("NotBound: " + exc.toString());

}

}

}

**policy.all**

grant {

permission java.security.AllPermission;

};

**Output:**

# Compile Java files

$ javac SampleServer.java

$ javac SampleServerImpl.java

$ javac SampleClient.java

# Start RMI registry

$ start rmiregistry

# Generate stubs using rmic

$ rmic SampleServerImpl

# Run the server

$ java -Djava.security.policy=policy.all SampleServerImpl

Security manager installed.

Server waiting.....

# Run the client

$ java -Djava.security.policy=policy.all SampleClient

Security Manager loaded

Got remote object

1 + 2 = 3

**Conclusion:** The client was able to access the services provided by the server using Java RMI (Remote Method Invocation). First, the server created a remote object and registered it with the RMI registry using Naming.rebind(). The client then used Naming.lookup() to retrieve this remote object using the specified URL. Once the reference to the remote object was obtained, the client successfully invoked the sum() method, which executed on the server and returned the computed result. This process enables seamless communication between distributed applications, allowing remote method execution as if they were local.