Experiment : 1

Aim : . Implement multi-threaded client/server Process communication using RMI.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Theory

RMI provides communication between java applications that are deployed on different servers and connected remotely using objects called stub and skeleton. This communication architecture makes a distributed application seem like a group of objects communicating across a remote connection. These objects are encapsulated by exposing an interface, which helps access the private state and behavior of an

object through its methods.

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed applications in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton.

RMI Registry, also known as the Java RMI Registry or simply the RMI registry, is a simple naming service provided by Java's Remote Method Invocation (RMI) technology. It acts as a central repository for registering and looking up remote objects in a distributed system.

The RMI registry serves as a directory where remote objects can be bound to names, allowing clients to locate and access these objects. It provides a mapping between the object's name (or a unique identifier) and the corresponding remote object reference.

Stub: This is a Java object that acts as an entry point for the client object to route any outgoing requests.

It exists on the client JVM and represents the handle to the remote object. If any object invokes a method

on the stub object, the stub establishes RMI by following these steps:

1. It initiates a connection to the remote machine JVM.

2. It marshals (write and transmit) the parameters passed to it via the remote JVM.

3. It waits for a response from the remote object and unmarshals (read) the returned value or exception,

then it responds to the caller with that value or exception.

Skeleton: This is an object that behaves like a gateway on the server side. It acts as a remote object with

which the client objects interact through the stub. This means that any requests coming from the remote

clients are routed through it. If the skeleton receives a request, it establishes RMI through these steps:

1. It reads the parameter sent to the remote method.

2. It invokes the actual remote object method.

3. It marshals (writes and transmits) the result back to the caller (stub).

Implementation:

serverInterface.java

import java.rmi.\*;

public interface ServerInterface extends Remote {

public String sayHello() throws RemoteException;

public int getNumClients() throws RemoteException;

public void addClient() throws RemoteException;

public void removeClient() throws RemoteException;

public String processMessage(String message) throws RemoteException;

}

Server.java

import java.rmi.\*;

import java.rmi.registry.\*;

import java.rmi.server.\*;

import java.util.Scanner;

public class Server extends UnicastRemoteObject implements ServerInterface {

private static final long serialVersionUID = 1L;

private static int numClients = 0;

public Server() throws RemoteException {

super();

}

public String sayHello() throws RemoteException {

return "Hello!";

}

public synchronized int getNumClients() throws RemoteException {

return numClients;

}

public synchronized void addClient() throws RemoteException {

numClients++;

}

public synchronized void removeClient() throws RemoteException {

numClients--;

}

public String processMessage(String message) throws RemoteException {

Scanner scanner = new Scanner(System.in);

System.out.println("\nReceived message: " + message);

System.out.print("Enter a message to send to the client: ");

String reply = scanner.nextLine();

return reply;

}

public static void main(String[] args) {

try {

Registry registry = LocateRegistry.createRegistry(1099);

Server server = new Server();

registry.rebind("Server", server);

System.out.println("\nServer started!");

} catch (Exception e) {

e.printStackTrace();

}

}

}

client.java

import java.rmi.\*;

import java.util.Scanner;

public class Client {

public static void main(String[] args) {

try {

Scanner scanner = new Scanner(System.in);

ServerInterface server = (ServerInterface) Naming.lookup("rmi://localhost/Server");

int clientId = server.getNumClients();

server.addClient();

System.out.println("\nClient " + clientId + " connected to the server.");

while (true) {

System.out.print("\nEnter a message to send to the server: ");

String message = scanner.nextLine();

String response = server.processMessage(message);

System.out.println("Server responded with: " + response);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

//MyClient.java

import java.util.Scanner;

import java.rmi.\*;

public class MyClient

{

public static void main(String args[])

{

try

{

Concat stub=(Concat)Naming.lookup("localhost");

Scanner input = new Scanner(System.in);

String x,y;

System.out.println("Enter String 1: ");

x = input.nextLine();

System.out.println("Enter String 2: ");

y = input.nextLine();

System.out.println(stub.concat(x,y));

}

catch(Exception e){}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//MyServer.java

import java.rmi.\*;

import java.rmi.registry.\*;

public class MyServer

{

public static void main(String args[])

{

try

{

Concat stub=new ConcatRemote();

Naming.rebind("localhost",stub);

}

catch(Exception e)

{

System.out.println(e);

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//ConcatRemote.java

import java.rmi.\*;

import java.rmi.server.\*;

public class ConcatRemote extends UnicastRemoteObject implements Concat

{

ConcatRemote()throws RemoteException

{

super();

}

public String concat(String x,String y)

{

return x+y;

}

}

//Concat.java

import java.rmi.\*;

public interface Concat extends Remote

{

public String concat(String x,String y)throws RemoteException;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Output:**

Terminal 1:

preeti@ubuntu:~/Desktop/rmi$ javac \*.java

preeti@ubuntu:~/Desktop/rmi$ rmic ConcatRemote

Warning: generation and use of skeletons and static stubs for JRMP

is deprecated. Skeletons are unnecessary, and static stubs have

been superseded by dynamically generated stubs. Users are

encouraged to migrate away from using rmic to generate skeletons and static

stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

preeti@ubuntu:~/Desktop/rmi$ rmiregistry

Terminal 2:

preeti@ubuntu:~/Desktop/rmi$ java MyServer

Terminal 3:

preeti@ubuntu:~/Desktop/rmi$ java MyClient

Enter String 1:

KBT

Enter String 2:

COE

KBTCOE

preeti@ubuntu:~/Desktop/rmi$



