

Institution of Technology

School of Computing

Department of Information Technology and computer science

Course Name: Introduction to Distributed System

Instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ [Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](mailto:yimer@wku.edu.et)

Lab 2: Remote Object Invocation using Java RMI

Objectives: After completing this lab. you will be able to:

• Demonstrate on how to implement remote method invocation using Java RMI.

• Definine remote interfaces, provide their implementation in a server computer, register the remote object using java registry and finally make remote method invocations from the client computer or programs.

• The following are the primary objectives of this lab session:

– Understanding the basics concepts of remote method invocations.

– Discussion of how interfaces are defined for Java RMI programs.

– Be able to implement remote interface definitions.

– Be able to register remote objects using Java registry.

– Be able to make remote method invocations from the client

2.1 Creating Distributed Applications by Using Java RMI

Aim:Write a program for implementing remote invocation using Java RMI. Using RMI to develop a distributed application involves these general steps:

1. Designing and implementing the components of your distributed application.

2. Compiling sources.

3. Making classes network accessible.

4. Starting the application.

Designing and Implementing the Application Components

First, determine your application architecture, including which components are local objects and which components are remotely accessible. This step includes:

• Defining the remote interfaces. A remote interface specifies the methods that can be invoked remotely by a client. The design of such interfaces includes the determination of the types of objects that will be used as the parameters and return values for these methods.

2-1

• Implementing the remote objects. Remote objects must implement one or more remote interfaces. The remote object class may include implementations of other interfaces and methods that are available only locally. If any local classes are to be used for parameters or return values of any of these methods, they must be implemented as well.

• Implementing the clients. Clients that use remote objects can be implemented at any time after the remote interfaces are defined, including after the remote objects have been deployed.

The key components of the RMI architecture are:

• Server object interface: A sub-interface of java.rmi.Remote that defines the methods for the server object.

• Server implementation: A class that implements the remote object interface.

• Server object: An instance of the server implementation.

• RMI registry: A utility that registers remote objects and provides naming services for locating objects.

• Client program: A program that invokes the methods in the remote server object.

• Client stub: An object that resides on the client host and serves as a surrogate for the remote server object.

• Server skeleton: An object that resides on the server host, and communicates with the stub and the actual server object.

Steps in writing RMI Applications

1. Defining the remote interfaces. This has the general form of a normal java interface except some additional characteristics. An object becomes remote by implementing a remote interface, which has the following char- acteristics:

• A remote interface extends the interface java.rmi.Remote.

• Each method of the interface declares java.rmi.RemoteException in its throws clause, in addition to any application-specific exceptions.

Example

import java.rmi.\*;

public interface AddInterface extends Remote {

public int sum(int n1, int n2) throws RemoteException;

}

2. Implementing the remote interface. We will write the implementation of the remote interface definition.

The implementation class should provide the implementation of the methods defined in the remote interface definition with constructors and main function. The things that need to be done in the implementation of the remote object are.

• Declare the remote interfaces being implemented

• Define the constructor for the remote object

• Provide an implementation for each remote method in the remote interfaces

Example

import java.rmi.\*;

import java.rmi.server.\*;

public class Add extends UnicastRemoteObject implements AddInterface {

public Add() throws RemoteException {

super();

}

public int sum(int n1, int n2) throws RemoteException {

return n1 + n2;

}

}

UnicastRemoteObject is a convenience class, defined in the RMI public API, that can be used as a superclass for remote object implementations. The superclass UnicastRemoteObject supplies implementations for a number of java.lang.Object methods (equals, hashCode, toString) so that they are defined appropriately for remote objects.

3. Create a main method for the server or a separate class that will create an object of the remote object and register it to the RMI registry. You need to include the java.rmi.Naming class to be able to register objects to the RMI registry.

public static void main(String args[]) {

try {

Naming.rebind("rmi://localhost:1099/Add", new Add()); System.out.println("Server is connected and waiting for the client");

} catch (Exception e) {

System.out.println("Server could not connect: " + e);

}

}

4. Implementing the clients. Clients that use remote objects can be implemented at any time after the remote interfaces are defined, including after the remote objects have been deployed. They simply make requests to access remote objects. The clients should locate the remote object and perform binding before they can invoke methods so they will need to contact the RMI registry to locate the remote object.

import java.rmi.Naming;

public class AddClient {

public static void main(String args[]) {

try {

AddInterface ai = (AddInterface) Naming.lookup("rmi://localhost:1099/Add"); System.out.println("The sum of 2 numbers is: " + ai.sum(10, 2));

} catch (Exception e) { System.out.println("Client Exception: " + e);

}

}

}

5. Compile the java files: compile all java files using java compiler (javac)

6. Run the Java RMI registry service: In a new command window run the rmi registry. Make sure that the RMI

registry is run from within the directory in which the remote object (RMI server) is implemented otherwise you

will get the error ClassNotFoundException.

:>start rmiregistry

7. Run the RMI server: Use the java runtime to execute the RMI server

:>java AddServer

8. Run the RMI client: User the java runtime to execute the RMI Client

:>java AddClient

RMI works as follows:

1. A server object is registered with the RMI registry.

2. A client looks through the RMI registry for the remote object.

3. Once the remote object is located, its stub is returned in the client.

4. The remote object can be used in the same way as a local object. Communication between the client and the server is handled through the stub and the skeleton.

Practical 1:

Write RMI program to implement a simple calculator. The remote object should provide functions that perform addition, subtraction, division and multiplication. The client should provide a menu based interface in which the user can enter the operation to be performed and the numbers in which the operation is going to be performed. The functions that need to be implemented in the remote object are:

1. float sum(float, float): for performing addition.

2. float sub(float, float): for performing subtraction.

3. float div(float, float): for performing division.

4. float mult(float, float): for performing multiplication.

Procedures:

Step 1: Write the Remote Interface

import java.rmi.\*;

public interface CalcInterface extends Remote {

public float add(float n1, float n2) throws RemoteException; public float sub(float n1, float n2) throws RemoteException; public float div(float n1, float n2) throws RemoteException; public float mult(float n1, float n2) throws RemoteException;

}

Step 2: Write the Remote Interface implementation (RMI Server), create a remote object and bind the object.

import java.rmi.\*;

import java.rmi.server.\*;

import java.rmi.Naming;

public class CalcServer extends UnicastRemoteObject implements CalcInterface {

public CalcServer() throws RemoteException {

super();

}

public float add(float num1, float num2) throws RemoteException {

return num1 + num2;

}

public float sub(float num1, float num2) throws RemoteException {

return num1 - num2;

}

public float div(float num1, float num2) throws RemoteException {

return num1 / num2;

}

public float mult(float num1, float num2) throws RemoteException {

return num1 \* num2;

}

public static void main(String args[]) {

try {

CalcServer cs = new CalcServer(); Naming.rebind("rmi://localhost:1099/Calculator", cs); System.out.println("Server is connected and waiting for the client ");

} catch (Exception e) {

System.out.println("Server could not connect: " + e);

}

}

}

Step 3: Write the RMI Client.

import java.rmi.Naming;

import java.util.\*;

public class CalcClient {

public static void main(String args[]) {

try {

Scanner in = new Scanner(System.in);

int choice;

float num1, num2, result = 0; CalcInterface cs

= (CalcInterface) Naming.lookup("rmi://localhost:1099/Calculator");

do {

System.out.println("\n\t\t Remote Calculator \n\t"); System.out.println("Enter 1 for Addition\n\t"); System.out.println("Enter 2 for Subtraction\n\t"); System.out.println("Enter 3 for Division\n\t"); System.out.println("Enter 4 for Multiplication \n\n\t");

System.out.println("Enter 5 to Exit \n\n\t"); System.out.print("Enter your choice : ");

choice = in.nextInt();

if (choice >= 1 && choice <= 4) { System.out.print("\n\tEnter first number : "); num1 = in.nextFloat(); System.out.print("\n\tEnter second number : "); num2 = in.nextFloat();

switch (choice) {

case 1:

result = cs.add(num1, num2);

break;

case 2:

result = cs.sub(num1, num2);

break;

case 3:

result = cs.div(num1, num2);

break;

case 4:

result = cs.mult(num1, num2);

break;

}

System.out.print("\n\n\tThe result of the operation is : " + result);

}

} while (choice != 5);

System.out.println("\n\n\tProgram Exiting..............");

} catch (Exception e) { System.out.println("Client Exception: " + e);

}

}

}

Step 4: Compile both the client and the server programs in the command line.

:> javac CalcServer.java

:> javac CalcClient.java

Step 5: Open a command prompt, go to the directory where the RMI server is located and start the RMI registry service.

:> start rmiregistry

Step 6: Run the RMI server in a new command prompt.

:> java CalcServer

Step 7: Run the RMI Client in a new command prompt, and test the program if it works properly.

:> java CalcClient

Exercise:

1. From the previous program, try to check if the server can handle more than one client at any given time.

2. Write a Java RMI Application which enables the client and server communicate by sending text messages.

It will be used to enable the server and client implement a simple chat application. They communicate by sending and receiving string values or messages.