Practical No. 8: Program to create a Client/Server application using RMI.

I. Practical Significance:

ODBC isn't appropriate for direct use from the Java programming language because it uses a C interface. The JDBC API was modeled after ODBC, but, because JDBC is a Java API, it offers a natural Java interface for working with SQL. JDBC is needed to provide a "pure Java" solution for application development.

II. Relevant Program Outcomes (POs)

PO1: Basic and Discipline Specific knowledge

PO2: Problem analysis

PO3: Design/ development of solutions

PO4: Engineering Tools, Experimentation and Testing

III. Competency and Practical skills

To develop Dynamic web Application

The practical is expected to develop the following skills:

3. Able to apply the JDBC to create table , and insert data in a table.
4. Able to demonstrate the use of various JDBC driver and tier application.

IV. Relevant Course Outcome(s)

Develop applications for Remote Method Invocation (RMI).

V. Practical Outcome (PrOs)

Program to create a Client/Server application using RMI.

VI. Minimum Theoretical Background

RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application 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 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**

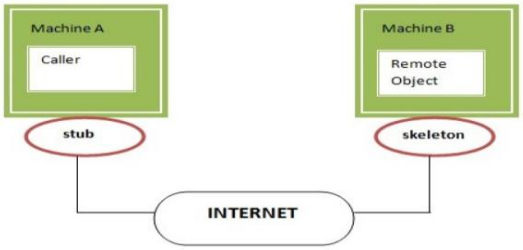
The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

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.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.

3. public void setFloat(int paramIndex, float value
4. public void setDouble(int paramIndex, double value
5. public int executeUpdate()
6. public ResultSet executeQuery

- (String SQL, int RSType, int RSConcurrency)
- prepareCall(String sql, int RSType, int RSConcurrency)
- Update methods



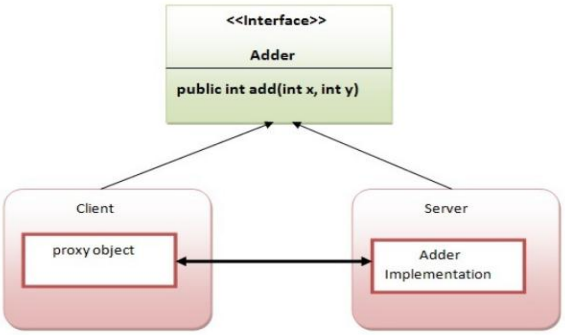
- **Applications of RMI**
  1. The application need to locate the remote method
  2. It need to provide the communication with the remote objects, and
  3. The application need to load the class definitions for the objects.
- **Java RMI Example**

The is given the 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

- **RMI Example**

  In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) create the remote interfaceFor creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2) Provide the implementation of the remote interface

- Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to Either extend the UnicastRemoteObject class, or use the exportObject() method of the UnicastRemoteObject class .In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder
{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects

```
rmic AdderRemote
```

4)Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It returns the reference of the remote object.
public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It binds the remote object with the given name.
public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;	It destroys the remote object which is bound with the given name.
public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;	It binds the remote object to the new name.
public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;	It returns an array of the names of the remote objects bound in the registry.

VII. Resources used (Additional)

Sr. No.	Name of Resource	Broad Specification	Quantity	Remarks (If any)

VIII. Practical Related Questions (students to attempt 3 to 4 questions. Min 2)

Note: Below given are few sample questions for reference. Teachers must design more such questions to ensure the achievement of identified CO.

1. What are the layers of RMI Architecture ?
2. What is the role of the java.rmi.Naming Class ?
3. What is the difference between using bind() and rebind() methods of Naming Class ?
4. Write steps involved to make work a RMI program ?
5. What is the role of Remote Interface in RMI ?

(Space for answer)

.....

.....

.....

.....

.....

.....

.....

.....