| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Practical Number:** 7 | |
| **Aim of Practical:**  Implementation of Remote Method Communication using JDBC and RMI. | | | |
| **DOP:** 12.10.23 | | **DOS:** 22.10.23 | |
| **CO Mapped:** CO4 | **PO Mapped:** - | **Faculty Signature:** | **Marks:** |

**Aim: Implementation of Remote Method Communication using JDBC and RMI.**

**Theory:**

Java Database Connectivity (JDBC): JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. Java JDBC Architecture is

1) JDBC Application- The JDBC application is in the topmost position of the JDBC

architecture. JDBC application is the data processing application that intends to access the data from the different data storage units.

2) JDBC API- The JDBC API plays a significant role in the JDBC architecture. The JDBC API ensures that a stable connection is established between the data storage unit and the JDBC application.

3) JDBC Manager- The JDBC manager takes care of selecting the appropriate driver manager to ensure that the driver software chosen supports the data storage unit's requirements to offer an uninterrupted connection.

4) JDBC Drivers-The JDBC driver is the crucial unit in the JDBC architecture. JDBC driver is the first layer of JDBC architecture that has direct contact with the data storage units.

5) Data Storage Units-Data storage units are the base of JDBC. The data storage unit is the place where all the data is kept accessible for the JDBC Applications.

Steps to Connect Java JDBC

Step 1: Import the database

Step 2A: Add the mysql connector to project

Step 2B: Loading the drivers- In order to begin with, you first need to load the driver or

register it before using it in the program. Registration is to be done once in your program.

Class.forName()- Here we load the driver’s class file into memory at the runtime. No need

of using new or create objects. The following example uses Class.forName() to load the

MYSQL driver as shown below as follows:

Class.forName(“com.mysql.jdbc.Driver”);

Step 3: Establish a connection using the Connection class object After loading the driver, establish connections via as shown below as follows:

Connection con = DriverManager.getConnection(url,user,password)

1) user: Username from which your SQL command prompt can be accessed.

2) password: password from which the SQL command prompt can be accessed.

3) con: It is a reference to the Connection interface.

4) url- Uniform Resource Locator. For eg. jdbc:mysql://localhost:3306/mysql

Step 4: Create a statement- Once a connection is established you can interact with the

database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.

Use of JDBC Statement is as follows:

Statement st = con.createStatement();

Step 5: Execute the query

Now comes the most important part i.e executing the query. The query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:

1) executeQuery() - The executeQuery() method of the Statement interface is used to

execute queries of retrieving values from the database. This method returns the object

of ResultSet that can be used to get all the records of a table.

2) executeUpdate(sql query)- The executeUpdate(sql query) method of the Statement

interface is used to execute queries of updating/inserting.

Step 6: Closing the connections- So finally we have sent the data to the specified location and now we are on the verge of completing of our task. By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection. It is as shown below as follows:

con.close();

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.

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 (marshalls) the result to the caller.

The 6 steps to write

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

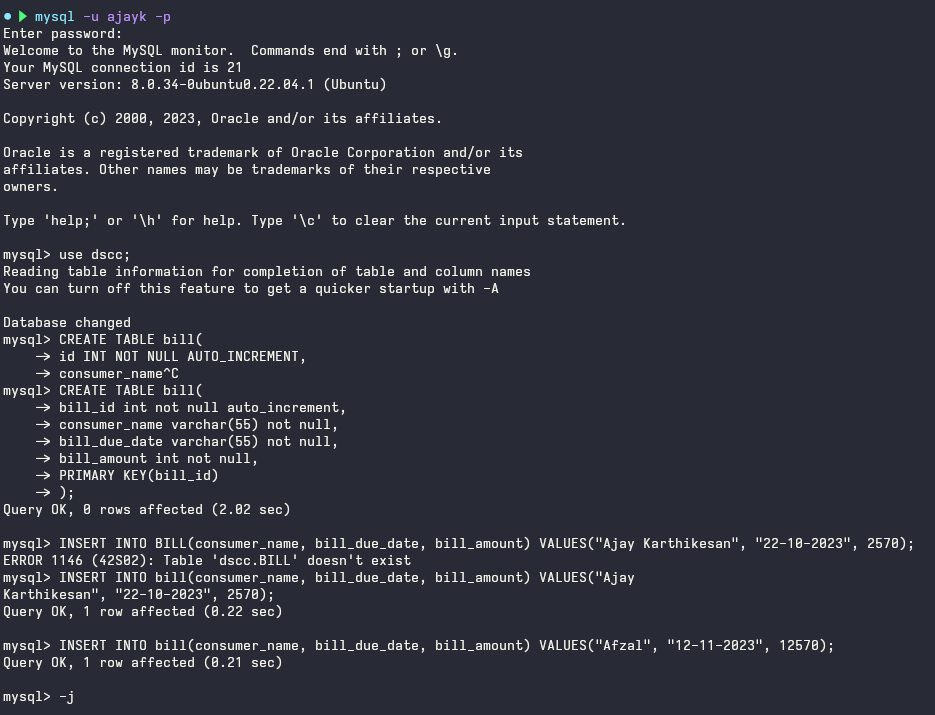
4. Start the registry service by rmiregistry tool

5. Create and start the remote application

6. Create and start the client application

**Using MySQL create Electric\_Bill database. Create table Bill. (bill\_id, consumer\_name, bill\_due\_date, bill\_amount) and retrieve the Bill information from the Electril\_Bill database.**

**DATABASE :**



**IDB.java:**

import java.rmi.\*;

public interface IDB extends Remote {

public String getData() throws RemoteException;

public String getData(int id) throws RemoteException;

}

**DBImpl.java:**

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.sql.Statement;

public class DBImpl extends UnicastRemoteObject implements IDB {

String str, str1;

public DBImpl() throws RemoteException {

}

@Override

public String getData(int id) throws RemoteException {

String URL = "jdbc:mysql://localhost:3306/dscc";

String UName = "ajayk";

String Pass = "0";

try {

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection(URL, UName, Pass);

PreparedStatement ps = con.prepareStatement("select \* from bill where bill\_id=?");

ps.setInt(1, id);

ResultSet rs = ps.executeQuery();

ResultSetMetaData rsmd = rs.getMetaData();

str = "";

str1 = "";

for (int i = 1; i <= rsmd.getColumnCount(); i++) {

str1 = str1 + rsmd.getColumnName(i) + "\t";

}

System.out.println();

while (rs.next()) {

for (int i = 1; i <= rsmd.getColumnCount(); i++) {

str = str + rs.getString(i) + "\t";

}

str = str + "\n";

}

} catch (Exception e) {

e.printStackTrace();

}

return (str1 + "\n" + str);

}

}

**Client.java:**

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.rmi.Naming;

public class Client {

public static void main(String[] args) {

String res = "";

try {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String url = "rmi://127.0.0.1/DBServerBill";

System.out.println("Retriving Bill Information....");

while (true) {

System.out.print("Enter choice:\n1. Get Bill By Id\n");

BufferedReader br1 = new BufferedReader(new InputStreamReader(System.in));

IDB id1 = (IDB) Naming.lookup(url);

res = id1.getData(Integer.parseInt(br1.readLine()));

System.out.println(res);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Server.java:**

import java.rmi.Naming;

public class Server {

public static void main(String[] args) {

try {

DBImpl di = new DBImpl();

Naming.rebind("rmi://127.0.0.1/DBServerBill", di);

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

} catch (Exception e1) {

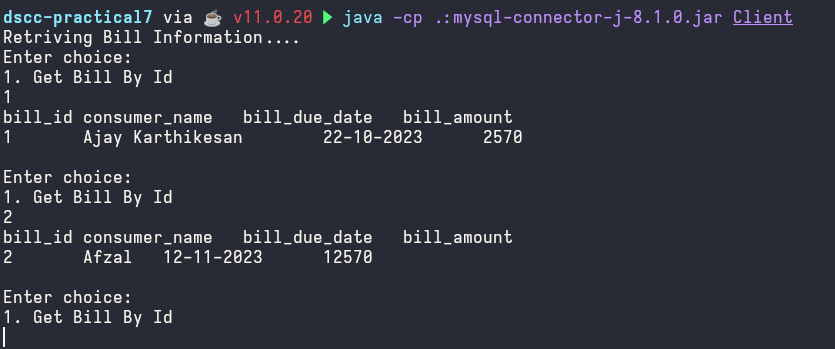
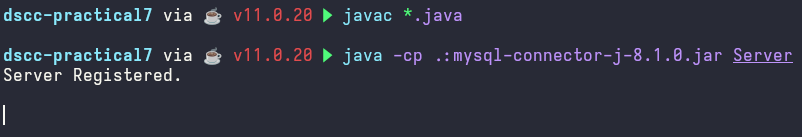
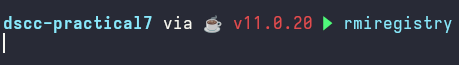
e1.printStackTrace();

}

}

}

**OUTPUT:**



**CONCLUSION:**

Thus, we successfully created a client-server-based application which helps us understand functioning of Remote Object Communication for database access