Module 1

Remote Method Invokation.

→ RMI

- It is a mechanism that allows an object residing in 1 system to access or invoke an object running on another JVM.
- RMI is used to build distributed application; it provides remote communication b/w Java programs. It is provided in the package: Java.rmi.

→ RMI Architecture of the RMI

- In RMI there are two programs, a server program and a client program.

(i) Inside the server program, a remote object is created and reference of that object is made available for the client

(ii) The remote client program requests the remote obj on the server and tries to invoke its method.

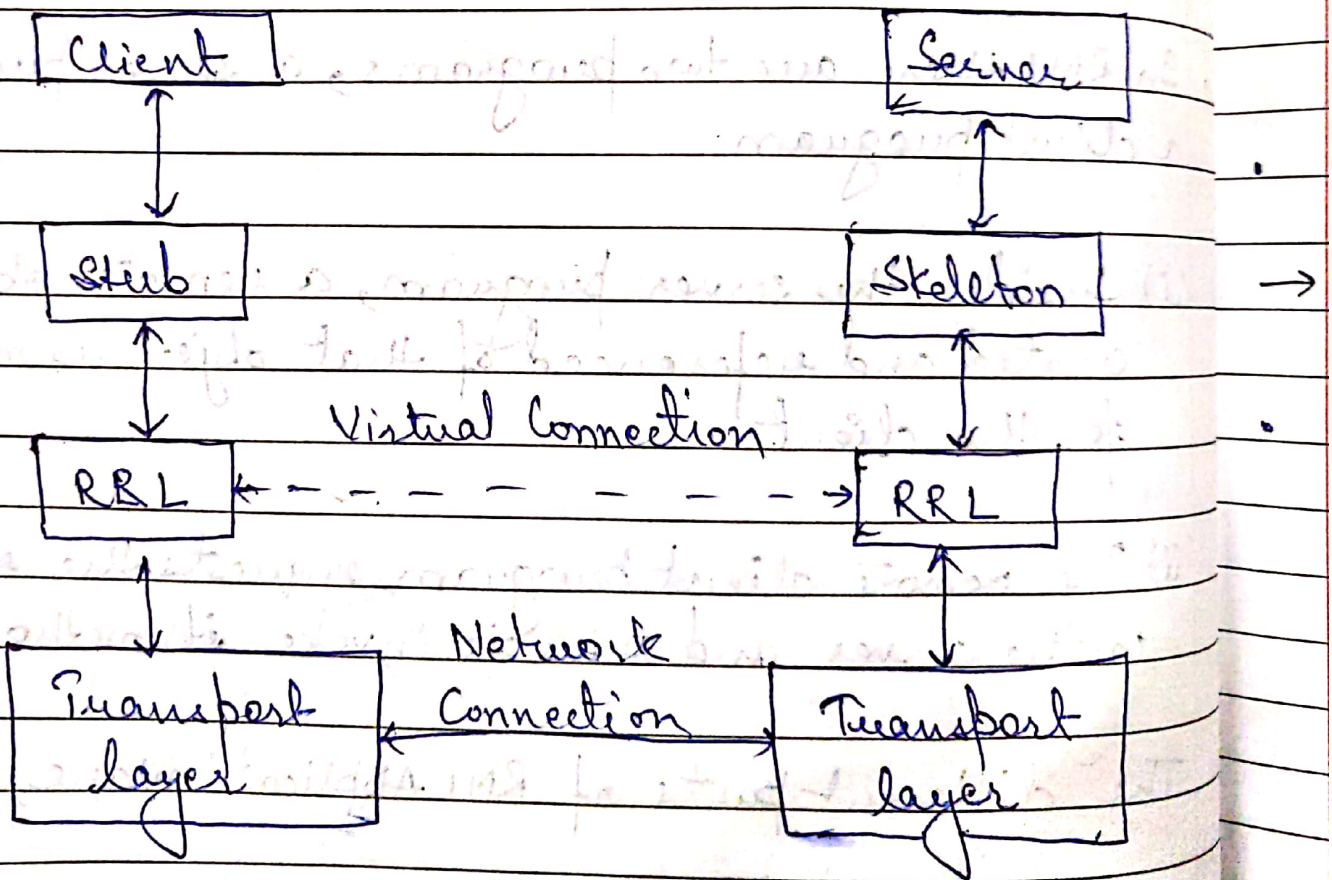
- The different parts of RMI Application are,

(i) Transport layer: This layer connects the client and the server. It manages the existing connection and also sets up new connection.

(ii) Stub :- A stub is representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway to the client program.

(iii) Skeleton :- This is the object which resides on the server side. Stub communicates with this skeleton to pass request to the remote object.

iv) RRL (Remote Reference layer) :- It is the layer which manages the references made by the client to the remote object.



→ RMI working

- When client makes the call to the remote obj. it is recieved by stub which extreamly eventually passes request to RRL.
- When client side RRL recieves the request. It invokes a method called invoke() of the objects remote ref. (RemoteRef). It passes requests to RRL of server side.
- The RRL of the server side passes the request to the skeleton which finally invokes the required object on the server.

The request is passed all the way back to the client.

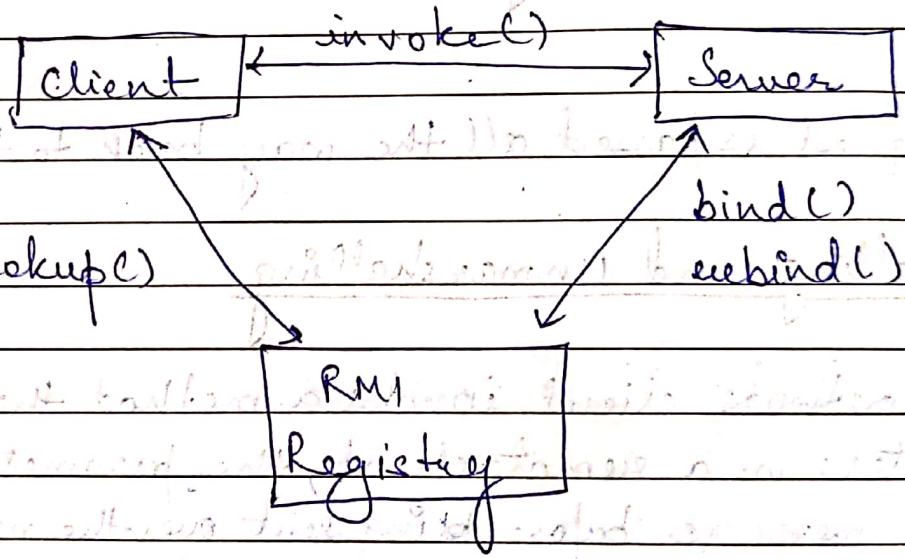
→ Marshalling and Unmarshalling

When methods client invokes a method that accepts parameters on a remote client, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it, In case of the parameters are objects, they are serialized this is called marshaling

At the server side, the packaged parameters are unbundled and then the required method is invoked. This is known as unmarshalling.

→ RMI Registry

- RMI Registry is a namespace on which all servers objects are placed. Each time the server creates an object, it registers this object with the RMI Registry (using bind() or rebind()). These are registered using a unique name known as bind name.
- To invoke a remote obj, the client needs a reference of that obj. At that time, the client fetches the object from the registry using its bind name (using lookup() method).



→ Goals of RMI

(i) Minimize the complexity of the application
(ii) To preserve type safety
(iii) Distributed garbage collection
(iv) Minimize difference b/w working with local and remote object.

→ RMI Programming

- Following is needed

(i) Remote interface
(ii) Implementation class
(iii) Server program
(iv) Client program

1) Remote Interface

```
import java.rmi.remote;
import java.rmi.remoteexception;

public interface Hello extends Remote
  {
    void print_msg () throws Exception;
  }
```

2.) Implementation Class (Remote obj)

```
Public class impExp implements Hello
  {
    public void print_msg ()
        {
          SOP ("This is example RMI program");
        }
  }
```

3) Server side program

```
Import java.rmi.registry.Registry;
"      "   "      "  , LocalRegistry;
"      "   "      ". RemoteException;
"      "   "      ". server. UnicastRemoteObject

Public class Server extends ImpExample {
  Public Server() { }
    PSVM (String args[])
    { try {
      → ImpExample obj = new implExample();
      → Hello stub = (Hello) UnicastRemoteObject.
                        exportObject(obj,
      → Registry registry = LocalRegistry.getRegistry();
      → registry.bind("Hello", stub);
      System.err.println("server exception" + e.toString());
    }
    Catch (Exception e)
    {
      System.err.println("Server Exception" + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Instance of ImpExample class exporting object using Unicast.

opening Registry

binding Registry with Remote object

Catching error if needed.

- We invoke the remote object from the client class.
- Create a remote object by installing the implementation class.

Kreator

- Export the remote object using method exportObject() of the UnicastRemoteObject class, which belongs to java.rmi.server.
- Get the RMI registry using getRegistry() method of the LocateRegistry class which belongs to class package java.rmi.registry
- Bind the remote object to the Registry using the bind() method of the class Registry. To this method, pass a String representing the bind name and the object exported.

4) Client Side program.

```
import java.rmi.registry.LocalRegistry;
import java.rmi.registry.Registry;
public class Client
{
  private client()
  {}
  psvm (String args[])
  {
    try
    {
      Registry registry = localRegistry.getRegistry(null);
      Hello stub = (Hello)registry.lookup("Hello");
      stub.printmsg();
      SOP("Remote Method Invoke");
    } Remote obj.
```

Creating the Registry

Looking up registry to find remote obj.

Creating Calling Func using

```
catch(Exception e)
{
System.err.println("ClientExcep"+
e.printStackTrace();
}
}
}
```

- Get the rmi registry using getRegistry() method of LocalRegistry class belonging to java.rmi.registry.
- Fetch the object from the registry using method lookup() of the class Registry of the java.rmi.Reg package.
- To this method, we need to pass a String value representing the bind name as parameter. This will return you the remote object.

The lookup() returns an object of type remote, down cast it to type Hello

⇒ Database

→ Types of database

1.) Relational DBMS
2.) NoSql
3.) Real time DB.
4.) Graph DB.

★ Carefully study all types of Key's done in DBMS for interview and also java exam.

⇒ What is PLSQL and stored procedure? (Homework)

★ Entry9india.com.

→ Creating Array in Java

Syntax:→ data type [] array_name = new datatype (size);

eg ⇒ int [] A = new int (10)

→ Creating Array of objects

Syntax:

Class-name [] array-name = new class-name (size);

eg ⇒ A [] abc = new A (10);

if we use as for loop we make a lot of objects.

Kreator

⇒ Array List

- It is a part of util library
  java.util.Arraylist.
- Syntax to declare is,

  wrapper class of int

  Arraylist<Integer> xyz = new Arraylist<Integer>();

- To get the size of Arraylist,

  SOP(xyz.size());

- We can add data as;

  xyz.add(new Integer(10));

  adds objects to array simultaneously.

  This creates an obj of 10 and sets it to heap. As we are using wrapper class we need to use object not direct int

  also,

  xyz.add(10); ← Also does the same as the above just in background

To check that if is added or not as,

if(xyz.contents(10));

Checks if present or not

Returns 1 (returns 0)

To check the index of added element;

int p = xyz.indexOf(x); ← int x = new Integer(10);

returns index of the obj reference in Arraylist.

To check if empty arraylist or not

if(xyz.isEmpty());

checks if empty or not

Enhanced for loop

```
class A
{ int a;
A(int p)
{
  sop(p); x = p;
}
void mup()
{
  sop(a)
}
Public Static Void main(String args[]){
Array list <A> xyz = new Arraylist<A>();
for(i=0; i<10; i++)
{
  xyz.add(new A(i));
}
for(A p : xyz)
```

enhanced for loop, it will take p of xyz which belongs to A class.

Creator

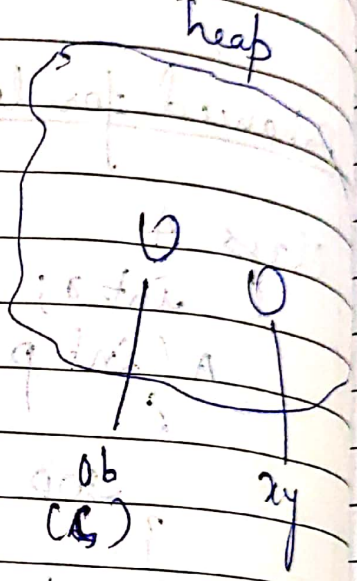
```
    }
        p.mup( );
    }
}
```

Output

0 1 2 3 4 5 6 7 8 9 (p)
0 1 2 3 4 5 6 7 8 9 (x)

⇒ Lifetime of an object

```
Class A
{
    int x, y;
    void xyz ( )
    {
        int l, n;
        sop("x+" " +l);
    }
}
Class ABC
{
    Psvm (String args[])
    {
        A ob = new A ( );
        A c = ob;
        A xy = new A ( );
        A ob = null;
    }
}
```

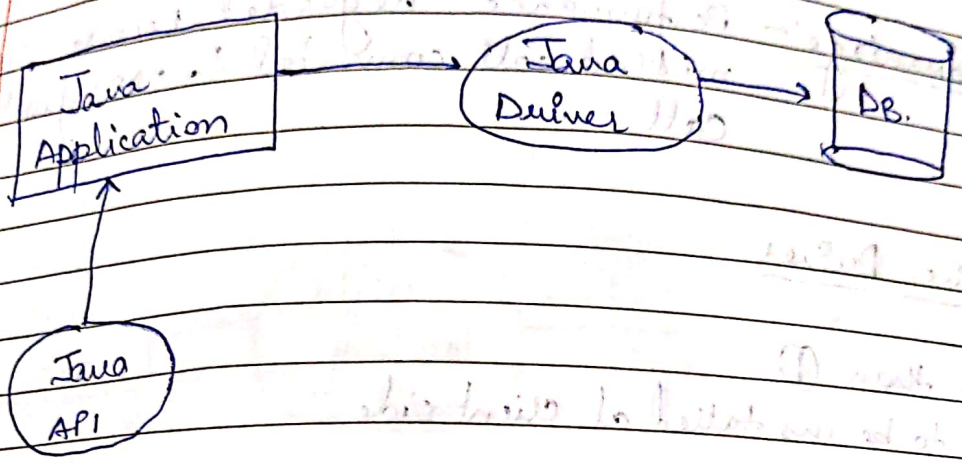


ob now also refers to c

No object is ready for garbage collection as these

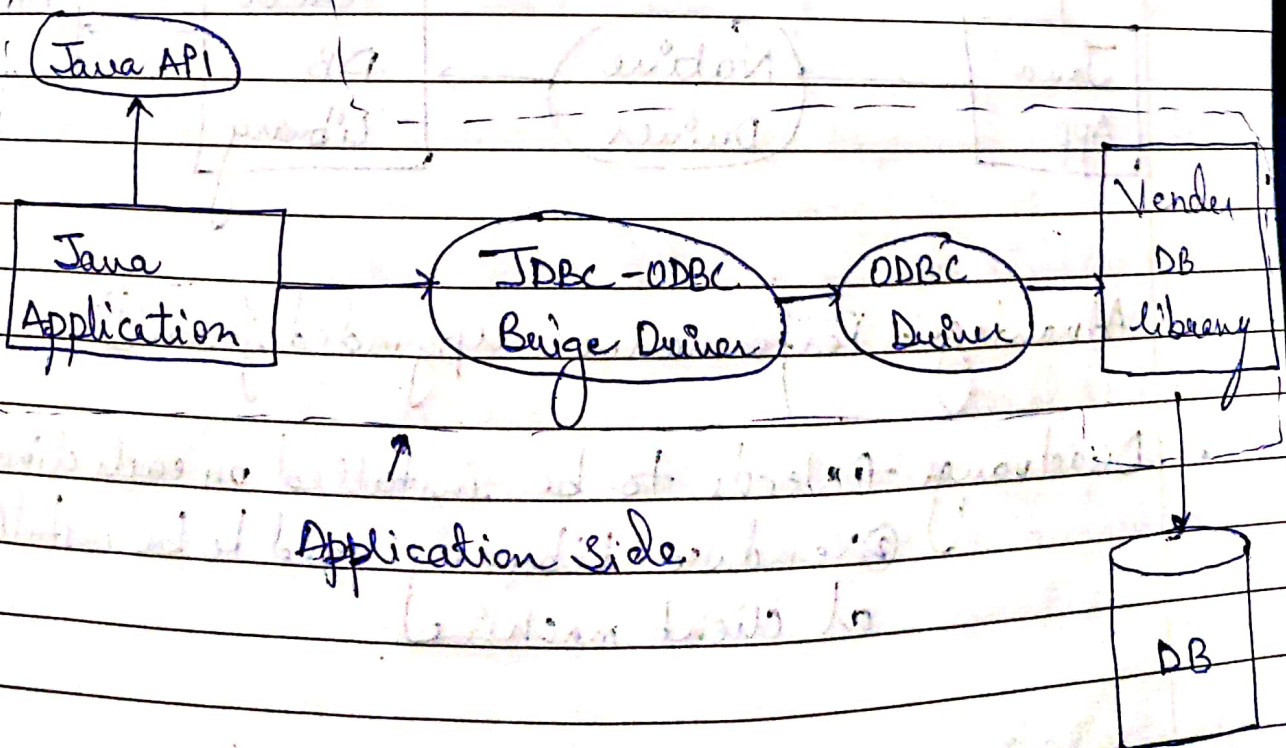
reference of C, object can only be destroyed when all the references are removed.

⇒ JDBC (Java Database Connectivity)



→ JDBC Drivers

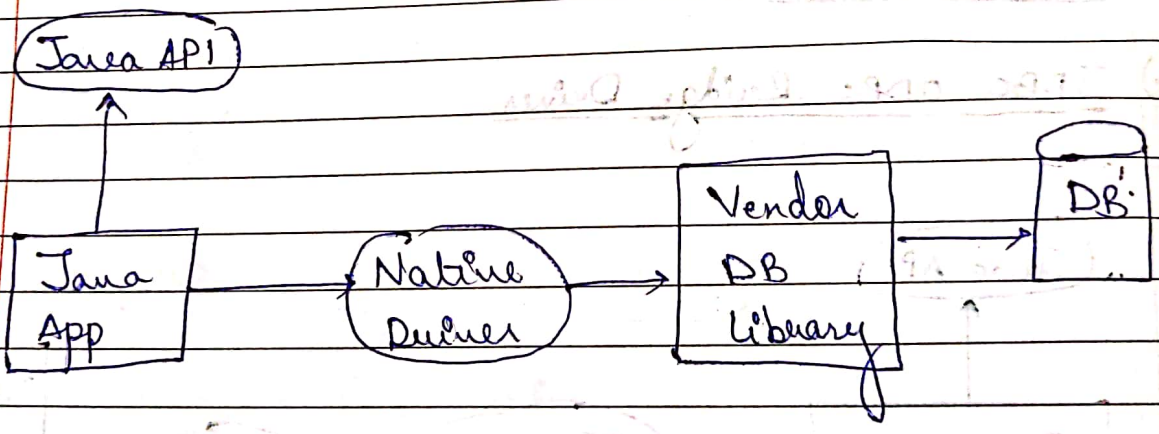
1.) JDBC ODBC Bridge Driver



- Takes a lot of time as Java API call needs to be converted to C API for ODBC
- Advantage :- easy to use
  Can be connected to any DB
- Disadvantage :- Performance degraded because JDBC method call converted to ODBC function call

2) Native Driver

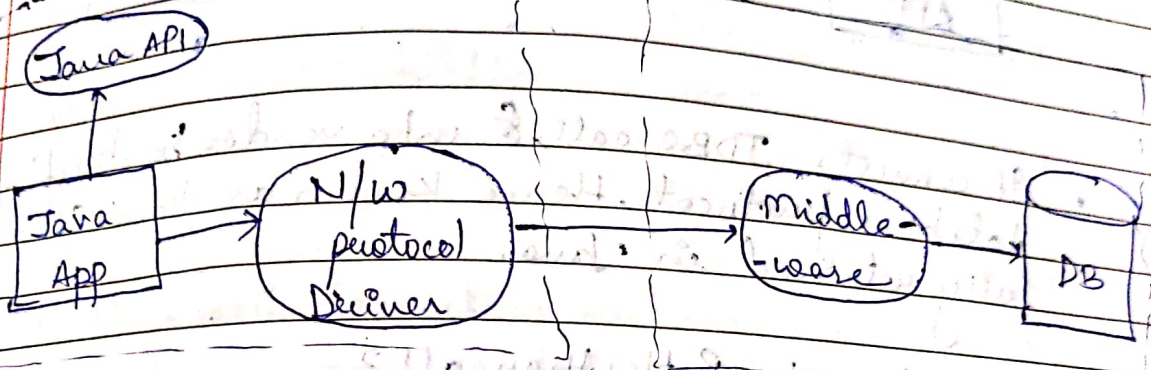
- Faster than (1)
- Needs to be installed at client side.



- Advantage :- Performance upgraded from (1)
- Disadvantage :- (1) Needs to be installed on each client m/c
  (2) Vendor library need to be installed at client machine

3) Network Drivers

- Network protocol Driver uses middleware or App server that converts JDBC call directly or indirectly into vendor specific database protocol. It is fully retained in Java



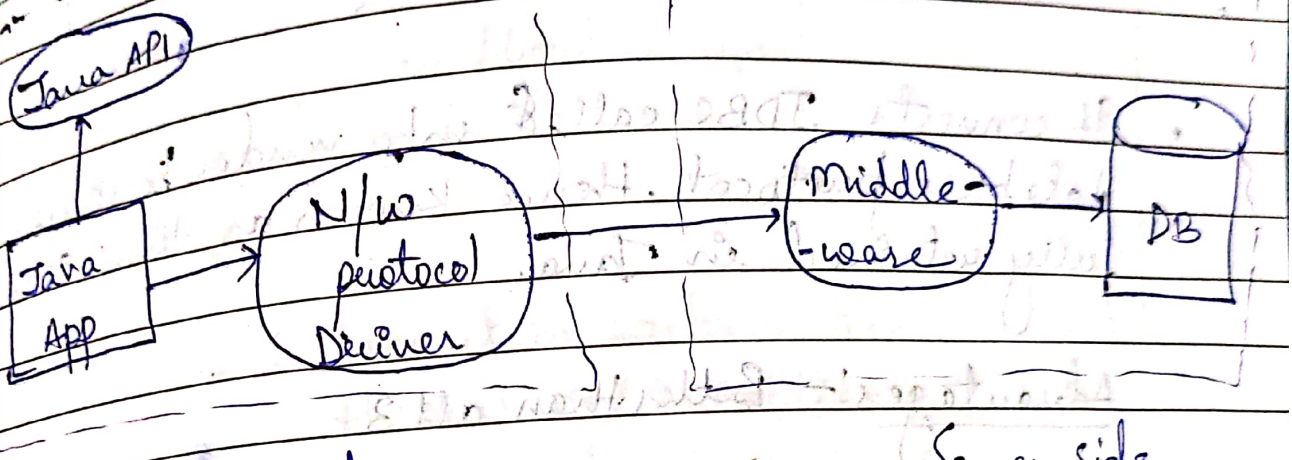
Client side Server side

Advantages :- No client side library is required because of App server that can perform many base line auditing, load balancing and login.

Disadvantages :- ① Network support required on client m/c
② Requires database specific codding to be done in middleware
③ Maitainance of N/w protocol drivers becomes costly because it requires database specific coding in middleware.

3) Network Drivers

. Network protocol Driver uses middleware on App server that converts JDBC call directly or indirectly into vendor specific database protocol. It is fully retained in Java



Client side Server side

Advantages :- No client side library is required because of App server that can perform many base line auditing, load balancing and login.

Disadvantages :- ① Network support required on client m/c

② Requires database specific codding to be done in middleware

③ Maitainance of N/w protocol drivers becomes costly because it requires database specific coding in middleware.

④ Execute Query

- Execute Query() method is used
- Syntax;

```
Resultset rs = stmt.executeQuery("select * from emp");
while(rs.next());   // gives true if not last element
{
    sop(rs.getInt(1) + " " + rs.getString(2)
        + " " + rs.getString(3));   // gives fields of diff DB types
}
```

⑤ Close Connection

- Close() method is used

Syntax:-

```
con.close();
```

⇒ RMI - GUI Application (RMI - continued)

- Here we will create an RMI application where a client invokes a method which displays a GUI window (Java Swing)

→ Defining the remote Interface

- We are defining a remote interface named Hello with a method named animation() in it.

(4) Execute Query

- Execute Query() method is used
- Syntax;

```
Resultset rs = stmt.executeQuery("select * from emp");
while(rs.next())          // gives true if not last element
{
    sop(rs.getInt(1) + " " + rs.getString(2)
        + " " + rs.getString(3));   // gives fields of diff DB table
}
```

(5) Close Connection

- Close() method is used

Syntax:-

```
con.close();
```

⇒ RMI - GUI Application (RMI - continued)

- Here we will create an RMI application where a client invokes a method which displays a GUI window (Int

→ Defining the remote Interface

- We are defining a remote interface named Hello with a method named animation() in it.

import java.rmi.Remote;
import java.rmi.RemoteException;
remote
public interface Hello extends Remote
{
    void animation() throws RemoteException;
}

→ Developing the Implementation Class

• Here we are trying to create window which display GUI contents, using JavaFx.

import JavaFx.animation.RemoteTransition;
import Javafx.application.Application;
import Javafx.event.EventHandler.

import Javafx.scene.*
import Javafx.stage.Stage;
import Javafx.util.Duration;

→ public Class FxSample extends Application implements Hello
{
@Override
public void start(Stage stage)
{
Box box = new box();
box.setwidth(150.0);
box.setheight(150.0);
box.setdepth(100.0);

Implementing Remote interface

Drawing box

properties of box

Creator

box.setTranslateX(350); ← position of box
box.setTranslateY(150);
box.setTranslateZ(50);

Setting text → Text text = new Text("Type letter to rotate box and click to stop");

Setting font, colour & position → text.setFont(Font.font(null, FontWeight.BOLD, 15));
text.setFill(Colour.CRIMSON);
text.setX(20);
text.setY(50);

Setting material of box → PhongMaterial material = new PhongMaterial();
material.setDiffuseColor(Color.DARKSLATEBLUE);
box.setMaterial(material);

Rotation Animation to the box → RotateTransition rotateTransition = new RotateTransition();
rotateTransition.setDuration(Duration.millis(1000));
rotateTransition.setNode(box);
rotateTransition.setAxis(Rotate.Y_AXIS);
rotateTransition.setByAngle(360);
rotateTransition.setCycleCount(50);
rotateTransition.setAutoReverse(false);

Creating textfield → TextField tf = new TextField();
tf.setLayoutX(80);
tf.setLayoutY(100);

Handling key events → EventHandler<KeyEvent> eventHandler Textfield = new EventHandler<KeyEvent>();
{

Kreator

Scanned by CamScanner

```
@Override
public void handle (KeyEvent event)
{
    rotateTransition.play();
}
};
```

Adding an event handler to next field

```
textfield.addEventHandler(KeyEvent.KEY_TYPED, eventhandler
                                                TextField);
```

Handling mouse events (Read from keyboard)

```
EventHandler<JavaFx.scene.input.MouseEvent.MouseClicked,
                                        eventhandlerBox);
Group root = new root(box, textfield, text);
Scene scene = new Scene (root, 600, 300);
```

Setting Camera.

```
PerspectiveCamera camera = new PerspectiveCamera
                                        (false);
Camera.setTranslateX (0);
"      "      "  Y (0);
"      "      "  Z (0);
scene.setcamera (camera);
```

Setting Stage

```
Stage.setTitle ("Event handler Example");
stage.setScene (scene);
Stage.show ();
```

Displaying contents of Stage

```
public void animation ()
{
    launch ();
}
```

Client side and server side programs remain same as in RMI coding.

Java RMI → Database Application

This is how we a client program invokes retrieves the record of a table in Mysql db residing on server.

Assuming we have a table student_data in the db having details,

ID	Name	Branch	%	Email
1	Ram	IT	85	ramb22@gmail
2	Rahim	EEE	95	rahim123@gmail
3	Robert	ECE	90	robertb2@gmail

Creating Class Student

```
public class Student implements Java.io.Serializable
{
    private int id, percent;
    private String name, branch, email;
    public int getId()
    {
        return id;
    }
    public String getBranch()
    {
        return branch;
    }
```

```
public String/int getPercentage()
{ return percent; }

public string getname()
{ return name; }

public string email()
{ return email; }

public void setID(int id)
{ this.id = id; }

public void setName(String name)
{ this.name = name; }

public void setBranch(String Branch)
{ this.Branch = Branch; }

public void setpercent(int percent)
{ this.percent = percent; }

public void setemail(String Email)
{ this.email = email; }

}
```

Defining Remote Interface

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.*;
```

public Interface Hello extends Remote
{
public list<Students> getstudent() throws Exception;
}

→ Developing Implementation Class

import java.sql.*;
import java.util.*;

public class ImplExample implements Hello
{
public list<Students> getstudents() throws Excep
{
List<Students> list = new Arraylist<Stude
String JDBC_DRIVER = "com.mysql.Idbc.
String DB_URL = "Jdbc:mysql://localhost:3306
String USER = "myuser";
String PASS = "password";

Connection conn = null;
Statement stmt = null;

Class.forName("com.sql.Jdbc.Drivers");
SOP("Connecting to selected db");
Conn = DriverManager.getConnection(DB_URL,
SOP("Connected Succesfully");

SOP("creating statement");

```
stmt = conn.createStatement();
String sql = "SELECT * from StudentData";
ResultSet rs = stmt.executeQuery(sql);

While (rs.next())
{
    int id = rs.getInt("id");
    String name = rs.getString("name");
    String Branch = rs.getString("Branch");
    int percentage = rs.getInt("percentage");
    String email = rs.getString("Email");

    Student stud = new Student();
    stud.setID(id);
    stud.setName(name);
    stud.setBranch(Branch);
    stud.setPercentage(percentage);
    stud.setEmail(Email);
    list.add(student);
    }
    rs.close();
    return list;
    }
}
```

* Server side program remains same as RMI Appt".

→ Client Program

```
import rmi.Registry.LocateRegistry;
import rmi.Registry.Registry;
import java.util.*;
```

Kreator

```
public class Client
{
  private Client()
  {
    psvm(String[] args) throws Excep...
    {
      try
      {
        Registry registry = locateRegistry.
                            getRegistry(...
        Hello stub = (Hello) registry.lookup
                     ("Hello");
        List<student> list = (List) stub.
                             getstudent();
        for (students s: list) {
        {
          SOP("ID: " + s.getId());
          SOP("name " + s.getname());
          SOP("branch " + s.getBranch());
          SOP("email " + s.getemail());
          SOP("percentage " + getpercentage...
        }
      catch (Exception e)
      {
        System.err.println("Client.exce...
                           + e.toString...
        e.printStackTrace();
      }
    }
```