Pabna University of Science and Technology

Practical Lab Report

Faculty of Engineering and Technology

Department of Information and Communication Engineering

Course Code : **ICE-3202**

Course Title : **Network Programming with Java Sessional**

Submitted By :
Mitu Rani Kundu
Roll : **190601**
Registration No : **1065326**
Session : **2018-19**
Department of **ICE , PUST**

Submitted To :
Md. Tofail Ahmed
Assistant Professor
Department of **ICE**
PUST

Submission Date:

Signature

INDEX

Experiment no: 01

Name of the Experiment: write a Java program to load and display multiple images into a swing container.

Theory:

Java swing is a part of Java foundation classes that is used to create window-based application. It is built on the top of AWT (Abstract window toolkit) API and entirely written in Java.

Unlike AWT, Java swing provides ~~classes~~ platform-independent and lightweight components.

The javax.swing package provides classes for Java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckBox, JMenu, JColorChooser etc.

Java Source Code:

```
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import javax.swing.*;
import javax.imageio.*;

public class Image extends JFrame
    {
        private JLabel label1;
        private JLabel label2;
        Image() throws IOException
            {
                SetLayout(new FlowLayout());
                File file = new File("D:\\network\\ICE-logo.jpg");
                BufferedImage image1 = ImageIO.read(file);
                ImageIcon imageIcon = new ImageIcon(image1);
                label1 = new JLabel(imageIcon);
                add(label1);

                File file2=new File("D:\\network\\Pust_logo.JPG");
                BufferedImage Image2 = ImageIO.read(file2);
                ImageIcon imageIcon2 = new ImageIcon(image2);
                label2 = new Label(imageIcon2);
                add(label2);
```

```
Public static void main(String[] args) throws IOException
    {
        Image gui = new Image();
        gui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        gui.setVisible(true);
        gui.setSize(500, 500);
        gui.setTitle("Image Program");
    }
}
```

Experiment no: 02

Name of the experiment: Suppose a restuarant sells pizza for $100, Bugers for $30 and Tea for $10. write a Java program for generating resturcant bills after ordering from a customer.

Source code:

```
import Javax.Swing.*;
Public class BillGeneration extends JFrame implements ActionLis-
                                                        tener
  {
    JLabel L;
    JCheckBox cb1, cb2, cb3;
    JButton b;
    BillGeneration()
      {
        L= new JLabel("Food ordering System");
        L.SetBounds(50,50, 300, 20);
        cb1 = new JCheckBox("pizza @ 100");
       cb1. SetBounds (100, 100, 150,20);
       cb2= new JcheckBox ('Burger @ 30");
       cb2. SetBounds (100,100,150,20);
```

```
cb3 = new checkbox ("Tea @ 10");

cb3.setBounds( 100, 200, 150, 20);

  b = new  JButton ("Order");

  b.setBounds(100, 200, 150, 20);

   b.addActionListener (this);

   add(l); add(cb1); add(cb2); add(cb3); add(b);

   setSize(400, 400);

   setLayout(null);

   setVisible(true);

    setDefaultCloseOperation (Exit_ON_CLOSE);

    }

   Public  void  actionPerformed(ActionEvent e)

        {

            float  amount=0;

             String msg="";

             if (cb1. isSelected())

                {

                  amout + = 100;

                  msg = "pizza : 100\n";

                 }
```

```
if (cb2.isSelected())
    {
       amount += 30;
      msg += "Tea : 10\n";
     }
     msg += "---------\n";
    JOptionPane.showMessageDialog(this, msg+ "Total : "+amount);
     }
   Public static void main(String[] args)
         {
            new BillGeneration();
          }
   }
```

Input:

Food Ordering system

☑ Pizza @100

☑ Burger @ 30

☐ Tea @10

Order

Output:

Message

pizza : 100

Burger : 30

Total 130.0

Ok

Experiment no: 03

Name of the experiment: write a Java program to create a Student registration form for ICE department including the fields "Name", "Roll" and "Department" in GUI.

Theory: GUI represents for Graphical User Interface. It actually represents that we see on our desktop which interacts with the user using a Graphical interface. Using GUI make it ~~easit~~ easy for the user to use the feature of an application.

In Java we can create our own desktop application using two GUI APIs that is AWT and Swing.

Program:

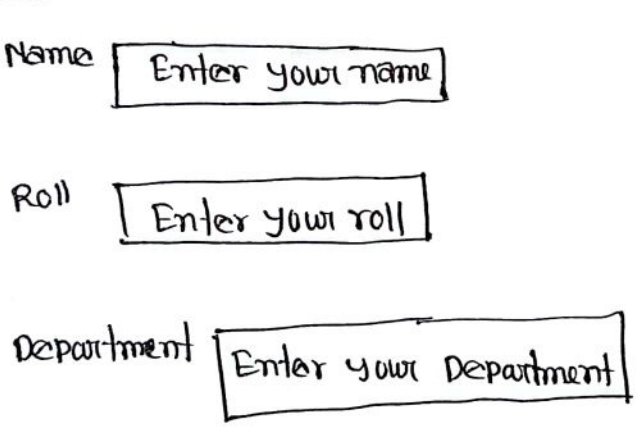
```
import java.awt.event.ActionEvent;
public class form implements ActionListener
{
    private static JLabel success;
    private static JFrame frame;
```

```
Private static JLabel label1, label2, label3;
Private static JPanel panel;
Private static JButton button;
Private static JTextField userText1, userText2, userText3;
Public static void main(String[] args)
    {
        frame = new JFrame();
        Panel = new JPanel();
        frame.setSize(400, 800);
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        panel.setLayout(null);
        label1 = new JLabel("Name");
        label1.setBounds(10, 10, 80, 25);
        Panel.add(label1);
        label2 = new JLabel("Roll");
        label2.setBounds(10, 60, 80, 25);
        Panel.add(label2);
        label3 = new JLabel("Department");
        label3.setBounds(100, 110, 80, 25);
```

```
userText1 = new JTextField ("Enter your roll");
userText1.setBounds(100, 10, 200, 25);
panel.add(userText1);

JTextFied userText2= new JTextField("Enter your roll");
userText2.setBounds(100, 60, 200, 25);
panel.add (userText2);

JTextfield userText3= new JTextfield("Enter your department");
  userText3.setBounds(100, 110, 200, 25);
  panel.add (userText3);
  button = new JButton("save");
  button.setBounds(150, 160, 80, 25);
  button.addActionListener(new Form());
  Panel.add(button);
  Succes = new JLabel("Not Submit Yet");
  Succes.setBounds(130, 210, 300, 25);
  Panel.add(succes);
```

```
frame.setVisible(true);
}
public void actionPerformed(ActionEvent e)
    {
        Succes.setText("Saved Succesfully");
    }
}
```

Input:



Output:

Experiment no: 04

Name of the experiment: write a Java program in GUI to develop a simple calculator that can calculate addition, subtraction, division and multiplication operations.

Program:

```
import java.awt.*;
public class ~~Jfa Clr~~ calculator extends JFrame implements
                                                ActionListener
{
    JButton b10, b11, b12, b13, b14, b15;
    JButton b[]=new JButton[10];
    int i,r, n1,n2;
    JTextField res;
    char op;
    public calculator()
    {
        super("calculator");
        setLayout(new BorderLayout());
        JPanel p= new JPanel();
```

```
p.setLayout (new GridLayout(4,4));

for(int i=0 ; i<=9 ; i++)
    {
      b[i]=new JButton(i+"");
      p.add (b[i]);
      b[i]. addAction_Listener (this);
      }
     b10 = new JButton("+");
     p.add(b10);
     b10. addActionListener (this);
     b11 = new JButton("-");
     p.add (b11);
     b11. addAction_Listener(this);
     b12 = new JButton("*");
     p.add(b12);
     b12. addAction Listener (this);
     b13 = new JButton("/");
     p.add (b13);
     b13. addActionListener (this);
     b14 = new JButton("=");
```

```
p.add(b14);
b14.addActionListener(this);

b15 = new JButton("C");
p.add(b15);
b15.addActionListener(this);

res = new JTextField(10);
add(p, BorderLayout.CENTER);
add(res, BorderLayout.NORTH);
setVisible(True);
Setsize(200,200);
}
public void ActionPerformed(ActionEvent ae)
    {
        JButton pb = (JButton) ae.getSource();
        if (Pb == b15)
          {
            r = n1 = n2 = 0;
            res.setText("");
          }
```

```
else if (pb == b14)
    {
        n2 = Integer.parseInt(res.getText());
        eval();
        res.setText("" + r);
    }
    else
    {
        boolean opf = false;
        if (pb == b10)
        {
            op = '+';
            opf = true;
        }
        if(pb == b11)
        {
            op = '-';
            opf = true;
        }
        if (pb == b12)
        {
            op = '*';
            opf = true;
        }
        if (pb = b11)
        { op = '-'; opf = true; }
```

```
if (Pb==b12)
  {
     op = '*';
     opf = true;
   }
if (Pb==b13)
    {
       op = '/';
       opf = true;
     }
if (Pb==false)
    {
      for(i=0; i<10; i++
        {
          if(Pb==b[i])
          {
             string t = res.getText();
              t += i;
              res.setText(t);
          }
        }
      }
  else
    {
      n1 = Integer.parseInt(res.getText());
      res.setText("");
```

```
}
}
}
int eval()
  {
    switch(op)
      {
        case '+' : r=n1+n2 ; break;
        case '-' : r=n1-n2 ; break;
        case '*' : r=n1*n2 ; break;
        case '/' ; r=n1/n2 ; break;
      }
    return 0;
    Public static void main (String args[])
        {
          new Calculator();
        }
    }
```

Output:

1 + 2 = 3			
0	1	2	3
4	5	6	7
8	9	+	-
*	/	=	c

Experiment no: 05

Name of the experiment: Suppose a multiple threads try to access the same resources and finally produce erroneous and unforeseen results. Write a Java program to solve this problem using object or method synchronization.

Theory:

A thread is similar to a program that has a single flow of control. On the other hands, threads are independent process that can be run simultaneously. It has

i) A beginning

ii) A body and an end

iii) Executes command sequentially.

Java enable us to use multiple flows of control in developing programs. Each flow of control may be thought of as separate tiny program known as thread that runs parallel. A program that contains multiple flows of control is known a multiple threaded program.

Source code:

```
class callme
 {
     void call(String msg)
         {
             system.out.printl("[ "+msg);
             try
               {
                 Thread.sleep(1000);
               } catch(InterruptedException e)
                   {
                     system.out.println(" Interrupted");
                     system.out.println("]");
                     }
         }

class caller implements runnable
     {
         String msg;
         callme target;
         Thread t;
         public caller(callme targ, String s)
             {
                 target = targ;
                 msg = s;
```

```
t = new Thread (this);
t.start();

  Public void run()
     {
        target.call(msg);
      }
    }
}

class synch
   {
      public static void main(String args[])
          {
              callme target = new callme();
              caller ob1 = new caller(target, "Hello");
              caller ob2 = new caller (target, "Synchronized");
              caller ob3 = new caller (target, "world");

            try
              {
                 ob1.t.join();
                 ob2.t.join();
                 Ob3.t.join();
              } catch(InterruptedException e){
```

```
        System.out.println("Interrupted");
      }
    }
}
```

Output:

```
[Synchronized [World [Hello]
]
]
```

Experiment no: 06

Name of the experiment: Write a client-server TCP socket program in Java that the server listens for connection requests and whatever message the client sends, the server converts it to uppercase and sends it back.

Theory:

Client: Initiates connection

- Retrieve data
- Displays data
- Responds to user input
- Requests more data

Examples of clients are

- Web Browser
- Chat program
- PC accessing files

Server: Responds to connection

- Receives request for data
- looks it up
- Delivers it

Examples of server:

- Web server
- Database server
- Domain Name server

TCP: The full form of TCP is Transmission Control Protocol. TCP is a connection-based protocol. It provides a reliable flow of data between two computers.

Source code:

server:

```
import java.net.*;
public class server1
 {
    public static void main( String args[]) throws IOException
          {
             ServerSocket s=new ServerSocket(2992);
```

```
System.out.println("waiting");

    Socket s1 = s.accept();

    InputStream s1In = s1.getInputStream();

    DataInputStream dis = new DataInputStream(s1In);

    String st = new String(dis.readUTF());

    System.out.println(st);

    OutputStream s1out = s1.getOutputStream();

    DataOutputStream dos = new DataOutputStream(s1.getOutput
                                                Stream());

    dos.writeUTF(st.toUpperCase());

    dos.close();
    s1out.close();
    s1.close();
  }
}
```

Output :

```
waiting
```

Client:

```
import java.net.*;

public class client1
  {
     public static void main(String args[]) throws IOException
       {
          Socket s1 =new socket("localhost", 2002);
          OutputStream s1out = s1.getOutputStream();
          DataOutputStream dos =new DatOutputStream (s1out);
          system.out.println(" write your menage ");
          scanner sc= new Scanner(System.in);
          String msg= sc.nextLine();
          dos.writeUTF(msg);
          InputStream s1In= s1.getInputStream();
          DataInputStream dis= new DataInputStream(s1In);
          String st= new String (dis.readUTF());
          system.out.println("output");
          system.out.println(st);
          s1.close s1.close();
       }
  }
```

Output:

write your menage: welcome to ICE

Output WELCOME TO ICE

Experiment no: 07

Name of the Experiment: Write a client-server UDP socket program in Java that the server listens for connection requests, and whatever message within 1024 bytes the client sends, the server converts it to uppercase and sends it back after 6 ms.

Theory:

UDP stands for User Datagram protocol. A protocol that sends independent packets of data called datagram, from one computer to another. In UDP, the arrival of datagrams is not guarantees. UDP is not connection-based like TCP.

Source code:

UDP server:

```
import java.io.*;
Public class UDPserver
    {
        public static void main(string args[]) throws Exception
            {
                DatagramSocket serversocket=new DatagramSocket
                (9876);
```

```
byte[] receiveData = new byte[1024];

byte[] sendData = new byte[1024];

        while(true)
          {

            DatagramPacket receivePacket = new DatagramPacket(receiveData,
                                                receiveData.length);

            System.out.println("waiting.....");

            serverSocket.receive(receivePacket);

            System.out.println("Data received-----");

            String sentence = new String(receivePacket.getData());

            System.out.println("RECEIVED" + sentence);

            Thread.sleep(3000);

              InetAddress IPAddress = receivePacket.getAddress();

               int port = receivePacket.getport();

               String capitalizedSentence = sentence.toUpperCase();

                sendData = capitalizedSentence.getBytes();

                DatagramPacket SendPacket = new DatagramPacket
                    (SendData, sendData.length, IPAddress, Port);

                    serverSocket.send(SendPacket);
        } } }
```

Output:

waiting....

UDP client:

```
import java.io.*;

Public class UDPclient
    {
      public static void main(String args[]) throws Exception
          {
            System.out.println("Write Something");

            BufferedReader infromUser=new BufferedReader(

                new InputStreamReader (System.in));

        DatagramSocket clientSocket = new DatagramSocket();
        Datagram Socke
        InetAddress IPAddress = InetAddress.getbyName("localhost");

          byte[] sendData = new byte[1024];

          byte[] receiveData = new byte [1024];

          String sentence = infromUser.readLine();

           sendData = sentence.inf sentence.getBytes();

          DatagramPacket sendPacket =new DatagramPacket
              (sendData, sendData.length, IPAddress, 9876);

            clientSocket.send(sendPacket);
```

```
DatagramPacket receivePacket = new DatagramPacket (receiveData,
                                    receiveData.length);
System.out.println("waiting....");
clientSocket.receive(receivePacket);
System.out.println(" Data Received ----");
String modifiedSentence = new String(receivePacket.getData());
System.out.println("FROM SERVER :"+ modifiedSentence);
clientSocket.close();
}
}
```

Output :

```
write Something
welcome to ica
waiting....
Data Received ----
FROM SERVER : WELCOME TO ICE
```

Experiment no: 08

Name of the experiment: Suppose we have an MS access database named ICE-PUST which has a table named by student with field (Name, Email and phone). Using this database answer the following question:

a) Write down a Java program to insert data into the Student table of the ICE-PUST database.

Theory:

Java Data Base Connectivity is the standard interface for communication between a Java application and SQL database. It allows a Java program to issue SQL statements and procen the results.

JDBC helps the programmers to write Java applications that manage three programming activities.

1) It helps us to connect to a data source, like a database.

2) It helps us in sending queries and updating statements to the database.

3. Retrieving and processing the results received from the database in terms of answering our query.

Source code:

```
import java.sql.*;
public class java_sql_db_insert_data
{
    public static void main(String[] args)
    {
    }
    try{
        Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
        Connection connection = DriverManager.getConnection
            ("jdbc:ucanaccess://D:\");
        System.out.println("Connected Succesfully");
        preparedStatement preparedStatement = connection.
            prepareStatement(" " + "INSERT
        INTO practice(Name, Email, phone) VALUES(?,??)");
        PreparedStatement.setString(1, "Rabbi");
        preparedStatement.setString(2, "Rabbi.ru@gmail.com");
```

```
preparedStatement.setString(3, "01722633140");

preparedStatement.executeUpdate();

System.out.println(" data inserted succesfully");

} catch(Exception e)
    {
        system.out.println("Error in connection");

        }
) }
```

Output:

Connected Succesfully

Data inserted Succesfully

Experiment no: 8(b)

Name of the experiment: Create a Java program to print all student records from the student table of ICE-PUST database.

Source code:

```
import java.sql.*;
public class Java_sql_db_show_data
  {
    public static void main(String[] args)
      {
        try
          {
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
            Connection Connection = DriverManager.getConnection("jdbc:
              ucanaccess://D:\Lab 18.accdb");
            PreparedStatement preparedStatement = Connection.preparedState-
                ment("select * from practice");
            ResultSet resultset = preparedStatement.executeQuery();
            System.out.println(resultset);
            System.out.println("Name " + "Email " + "Phone");
```

~~System.out.println (name+"~~

```
while ( resultset.next())
    {
        String name = resultset.getString ("name");
        string email = resultset.getstring ("Email");
        String phone = resultset.getString ("phone");
        system.out.println(name + " " + email + " " + phone);
        }
    } catch ( Exception e)
        {
            system.out.println("Error in connection");
        }
    }
}
```

Output:

Connected succenfully

Name	Email	phone
Jalal Ahmed	Jalal.ru@gmail.com	01557240628
Sofiq	sofiq@gmail.com	01557240628
Rabbi	rabbi.ru@gmail.com	01722633140

Experiment no: ~~9(a)~~ 9

Name of the Experiment: Consider an MS access database named Lab-18 which has a table named by Teacher with field (Name, Email and Phone). Using this database give the answer of this following questions.

a) Write down a Java program to create a GUI registration form according to Teacher table.

b) Create a Java program to insert data into Teacher table of the Lab-18 database from the GUI registration form which you have already created.

Objective: To create a GUI registration form according to a database. ~~table.~~

Theory:

Loading a JDBC Driver

- A JDBC driver is needed to connect to a database.
- Loading a driver requires the class name of the driver
  - JDBC-ODBC : Sun.jdbm.odbc.JdbcOdbcDriver
  - Oracle driver : Oracle.jdbc.driver.OracleDriver
  - MySQL : com.mysql.jdbc.Driver
  - UcanAccess : net.ucanaccess.jdbc.UcanaccessDriver

- Loading the driver class

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

Connecting to a database:

JDBC URL for a database

- Identifies the Database to be connected
- Consists of three parts: protocol, sub-protocol, subname

- The driverManager allows to connect to a database using the specified JDBC driver, database location, database name, username and password.

- It returns a connection object which can then be used to communicate with the database.

Source code:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;
import javax.swing.*;
public class java_sql_gui_1_simple extends JFrame implements ActionListen
```

```
{
    JTextField tf1, tf2, tf3;
    JLabel l1, l2, l3;
    JButton b1, b2, b3, b4;
    Connection con;
    ResultSet rs;
    Statement st;

Public Java_sql_gui_1_Simple
    {
        Super("PersonDetails");
        SetLayout(null);
        Label d = new Lable("Person Details");
        d.SetBounds(200,10,150,50);
        d.SetBackground(color.RED);
        add(d);
        l1 = new JLabel("Name");
        l1.SetBounds(100,100,100,50);
        add(l1);
        tf1 = new JTextField(" ");
        tf1.SetBounds(200,100,250,40);
        l2 = new JLabel("Email");
        l2.setBounds(100,150,200,50);
```

```
add(l2);
tf2 = new JTextField();
tf2.setBounds(200,150,250,40);
add(tf2);
l3 = new JLabel("Phone");
l3.setBounds(100,200,100,50);
add(l3);
tf3 = new JTextField();
tf3.setBounds(200,200,250,40);
add(tf3);
b1 = new JButton("show");
b1.setBounds(100,250,90,50);
add(b1);
b2 = new JButton("Insert");
b2.setBounds(200,250,90,50);
add(b2);
b3 = new JButton("Update");
b3.setBounds(300,250,90,50);
add(b3);
```

```
b4= new JButton("Delete");

b4.SetBounds(400, 250, 090, 50);

add(b4);

b1.addActionListener(this);

b2. addAction Listener(this);

b3. addActionListener(this);

b4. addActionListener(this);

 SetSize(600, 600);

 Setvisible(true);

 Connection();

 }

Public void connection()

      {

          try

            {

                Class.forName("net.ucanaccon.jdbc.UcanaccessDriver");

                con= Driver Manager.getConnection("jdbc:ucanaccess://D:\\");

                system.out.println("connected Successfully");

            st= con.createStatement(Result.set.TYPE_SCROLL_INSENSITIVE;

                    Resultset. CONCUR_UPDATABLE);
```

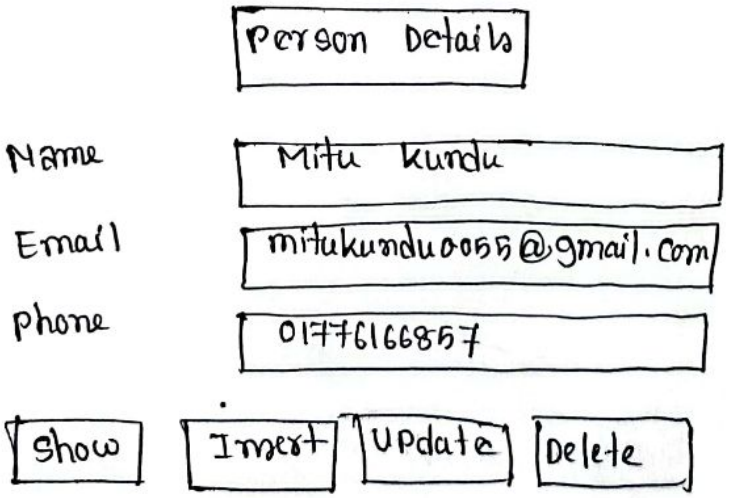
```
rs = st. executeQuery("SELECT * FROM practice");

System.out.println("connect");

} catch(Exception e)

    {

        System.out.println(e);

    }

}

Public void actionPerformed (ActionEvent e)

        {

            if (e.getSource() == b1)

                {

                    System.out.println("I am show");

                 try

                 {

                    if (rs.next())

                      {

                        tf1.setText( rs.getString("name");

                        tf2.setText( rs.getString("Email"));

                        tf3.setText (rs.getString( "Phone"));

                      } else

                      {

                        JOptionPane.showMessageDialog(null, "nodata");

                      } catch(Exceptin ex)
```

```
{

}}

if (e.getSource()==b2)
    {
        System.out.println("I am insert");
        String up1 = tf1.getText();
        string up2 =tf2.getText();
        String up3 = tf3.getText();
        try
        {
            rs.updateString("Name",up1);
            rs.updateString("Email", up2);
            rs.updateString("Phone", up3);
            rs.insertRow();
            System.out.println("one row is inserted");
        }
        catch(Exception ex)
        {
        }
    }

if (e.getSource()==b3)
    {
```

```
System.out.println("I am Update");

String up1 = tf1.getText();

String up2 = tf2.getText();

String up3 = tf3.getText();

try
  {
     rs.updateString("name", up1);
     rs.updateString("Email", up2);
     rs.updateString("phone", up3);
     rs.updateRow();
     System.out.println("One row is updated");
  }
  catch(Exception ex)
   {
   }
  }

if (e.getSource() == bq)
  {
    System.out.println("I am Delete");
      try
        {
          rs.deleteRow();
          system.out.println("one row is deleted");
```

```
    {
    }
  }
}

Public static void main (String a[])
        {
            neo - sql - gui - 1 - Simple();
        }
    }
```

~~Output~~ : Input :



Output :

Connected succesfully

Connect

I am Insert

One row is inserted