

Experiment No: 01

Experiment Name: Loading and Displaying Multiple Images in a Swing Container

Objectives:

1. To understand how to use Java Swing components to create a graphical user interface (GUI) application.
2. To learn how to load and display multiple images within a Swing container.

Theory:

Java Swing is a powerful library for creating GUI applications. It provides a range of components to build interactive and visually appealing interfaces. In this experiment, we will focus on loading and displaying multiple images using Swing components.

The JFrame class is used to create the main window of the application. We will create a JPanel to hold our images and use a JScrollPane to allow scrolling through the images. ImageIcon class will help us load and display the images.

Source Code:

```
import java.awt.FlowLayout;

public class Image extends JFrame {
    private final JLabel label1;
    private final JLabel label2;
    Image() throws IOException{
        setLayout(new FlowLayout());
        //image1 = new ImageIcon(getClass().getResource("ICE_logo.jpg")); for jdk old version(8,6 etc)
        File file = new
        File("ICE_logo.jpg");
        BufferedImage image1 = ImageIO.read(file);
        ImageIcon imageIcon = new ImageIcon(image1);
        label1 = new JLabel(imageIcon);
        add(label1);

        //image2 = new ImageIcon(getClass().getResource("Pust_logo.png")); for jdk old version
        File file2 = new File("Pust_logo.png");
        BufferedImage image2 = ImageIO.read(file2);
        ImageIcon imageIcon2 = new ImageIcon(image2);

        label2 = new JLabel(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.pack(); Fit image to the prescribe size of container
        gui.setTitle("Image Program");
    }
}
```

Output:



Experiment No: 02

Experiment Name: Suppose a restaurant sells Pizza for \$100, Burgers for \$30, and Tea for \$10. Write a Java program for generating restaurant bills after ordering from a customer

Objectives:

1. To write a Java program that can generate restaurant bills after ordering from a customer.
2. To understand the basic concepts of Java programming, such as variables, arrays, loops, and functions.
3. To be able to apply these concepts to solve real-world problems.

Theory:

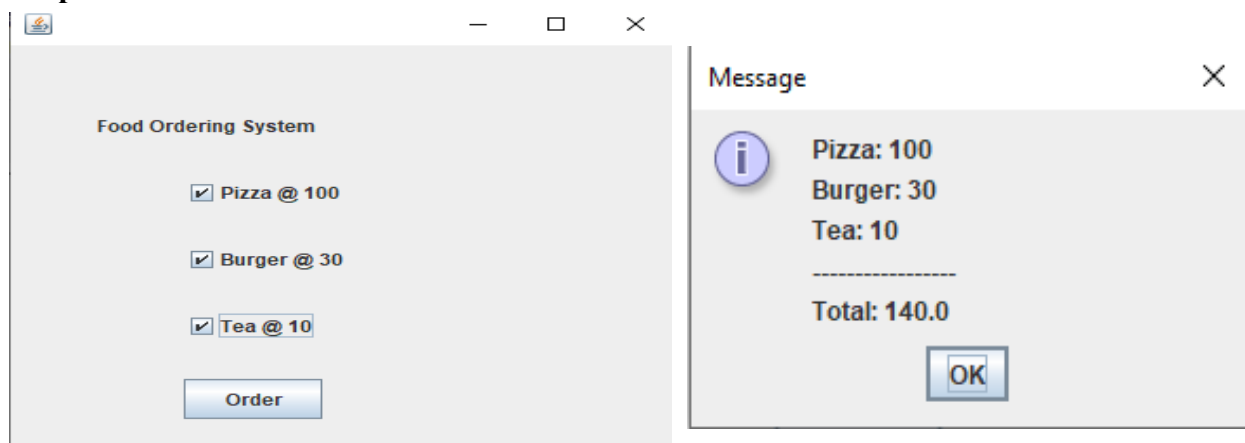
A restaurant bill is a document that lists the items ordered by a customer and their corresponding prices. It also includes the total amount due, including tax and tip. The Java program that we will write will take the customer's order as input and generate a restaurant bill as output. The program will use the following concepts:

- Variables: Variables are used to store data. In our program, we will use variables to store the customer's order, the prices of the items, and the tax and tip rates.
- Arrays: Arrays are used to store a collection of data. In our program, we will use an array to store the items in the customer's order.
- Loops: Loops are used to execute a block of code repeatedly. In our program, we will use a loop to iterate through the items in the customer's order and calculate the total price.
- Functions: Functions are used to group together related code. In our program, we will define a function to calculate the total price of the customer's order.

Source Code:

```
1 import javax.swing.*;
2
3 public class BillGeneration extends JFrame implements ActionListener{
4     JLabel l;
5     JCheckBox cb1,cb2,cb3;
6     JButton b;
7     BillGeneration(){
8         l=new JLabel("Food Ordering System");
9         l.setBounds(50,50,300,20);
10        cb1=new JCheckBox("Pizza @ 100");
11        cb1.setBounds(100,100,150,20);
12        cb2=new JCheckBox("Burger @ 30");
13        cb2.setBounds(100,150,150,20);
14        cb3=new JCheckBox("Tea @ 10");
15        cb3.setBounds(100,200,150,20);
16        b=new JButton("Order");
17        b.setBounds(100,250,80,30);
18        b.addActionListener(this);
19        add(l);add(cb1);add(cb2);add(cb3);add(b);
20        setSize(400,400);
21        setLayout(null);
22        setVisible(true);
23        setDefaultCloseOperation(EXIT_ON_CLOSE);
24    }
25
26    public void actionPerformed(ActionEvent e){
27        float amount=0;
28        String msg="";
29        if(cb1.isSelected()){
30            amount+=100;
31            msg+="Pizza: 100\n";
32        }
33        if(cb2.isSelected()){
34            amount+=30;
35            msg+="Burger: 30\n";
36        }
37        if(cb3.isSelected()){
38            amount+=10;
39            msg+="Tea: 10\n";
40        }
41        msg+="-----\n";
42        JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
43    }
44    public static void main(String[] args) {
45        new BillGeneration();
46    }
47 }
```

Output:



Experiment No: 03

Experiment Name: Write a Java Program to create a student registration form for ICE department including the fields "Name", "Roll", and "Department" in GUI.

Objectives:

1. To learn how to create a graphical user interface (GUI) application using Java Swing.
2. To understand the process of designing a student registration form with specific fields.
3. To practice capturing and processing user input through GUI components.

Theory:

A GUI (Graphical User Interface) is a type of user interface that uses graphical elements, such as buttons, text fields, and labels, to interact with the user. GUI programming is a complex topic, but it is essential for creating user-friendly applications.

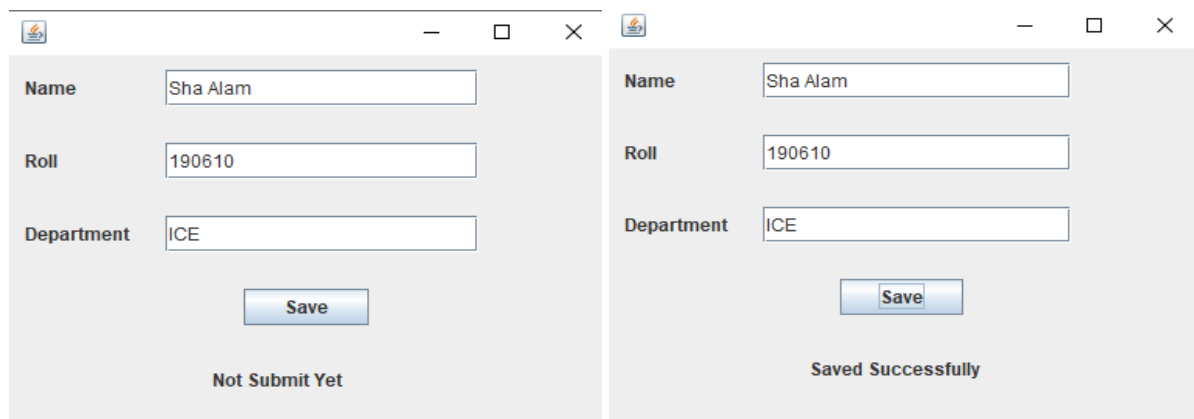
The Java Swing library provides a set of classes that can be used to create GUI applications in Java. In this experiment, we will use the Swing library to create a student registration form for the ICE department.

The form will have three text fields for the student's name, roll number, and department. It will also have a button to submit the form. When the user clicks the submit button, the program will validate the input and then print a message to the console confirming the registration.

Source Code:

```
1 import java.awt.event.ActionEvent;
2 public class Form implements ActionListener {
3     private static JLabel success;
4     private static JFrame frame;
5     private static JLabel label1, label2, label3;
6     private static JPanel panel;
7     private static JButton button;
8     private static JTextField userText1, userText2, userText3;
9     public static void main(String[] args) {
10         frame = new JFrame();
11         panel = new JPanel();
12         frame.setSize(400, 300);
13         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14         frame.add(panel);
15         panel.setLayout(null);
16         //Setting all Three Labels
17         label1 = new JLabel("Name");
18         label1.setBounds(10, 10, 80, 25);
19         panel.add(label1); label2 = new JLabel("Roll");
20         label2.setBounds(10, 60, 80, 25);
21         panel.add(label2);
22         label3 = new JLabel("Department");
23         label3.setBounds(10, 110, 80, 25);
24         panel.add(label3);
25         //Creating all Textfields
26         userText1 = new JTextField("Enter Your Name");
27         userText1.setBounds(100, 10, 200, 25);
28         panel.add(userText1);
29         JTextField userText2 = new JTextField("Enter Your Roll");
30         userText2.setBounds(100, 60, 200, 25);
31         panel.add(userText2);
32         JTextField userText3 = new JTextField("Enter Your Department");
33         userText3.setBounds(100, 110, 200, 25);
34         panel.add(userText3);
35         button = new JButton("Save");
36         button.setBounds(150, 160, 80, 25);
37         button.addActionListener(new Form()); //not use this keyword,
38         //because it called from main method, if we use constructor then we can use this keyword
39         panel.add(button);
40         success = new JLabel("Not Submit Yet");
41         success.setBounds(130, 210, 300, 25);
42         panel.add(success);
43         frame.setVisible(true);
44     }
45     @Override
46     public void actionPerformed(ActionEvent e) {
47         // TODO Auto-generated method stub
48         success.setText("Saved Successfully");
49     }
50 }
```

Output:



Experiment No: 04

Experiment Name: Write a Java Program in GUI to develop a simple calculator that can calculate addition, subtraction, division, and multiplication operations.

Objectives:

1. To write a Java program that can create a simple calculator with a graphical user interface (GUI).
2. To understand the basic concepts of Java GUI programming, such as labels, text fields, buttons, and event handlers.
3. To be able to apply these concepts to create a useful and user-friendly application.

Theory:

A GUI (Graphical User Interface) is a type of user interface that uses graphical elements, such as buttons, text fields, and labels, to interact with the user. GUI programming is a complex topic, but it is essential for creating user-friendly applications.

The Java Swing library provides a set of classes that can be used to create GUI applications in Java. In this experiment, we will use the Swing library to create a simple calculator with a GUI.

The calculator will have two text fields for the first and second numbers, and it will have four buttons for addition, subtraction, multiplication, and division. When the user clicks a button, the program will perform the corresponding operation on the two numbers and display the result in the text field.

Source Code:

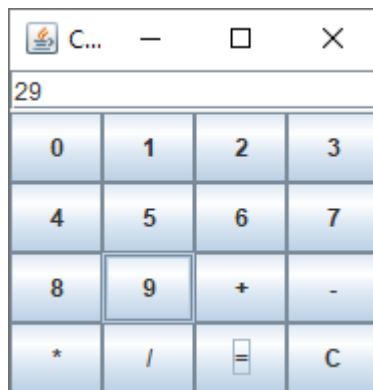
```
1 import java.awt.*;
2 public class Calculator extends JFrame implements ActionListener
3 {
4     JButton b10,b11,b12,b13,b14,b15;
5     JButton b[]=new JButton[10];
6     int i,r,n1,n2;
7     JTextField res;
8     char op;
9     public Calculator()
10    {
11        super("Calculator");
12        setLayout(new BorderLayout());
13        JPanel p=new JPanel();
14        p.setLayout(new GridLayout(4,4));
15        for(int i=0;i<=9;i++)
16        {
17            b[i]=new JButton(i+"");
18        }
19    }
20 }
```

```

20  b[i]=new JButton(i+"");
21  p.add(b[i]);
22  b[i].addActionListener(this);
23  }
24  b10=new JButton("+");
25  p.add(b10);
26  b10.addActionListener(this);
27  b11=new JButton("-");
28  p.add(b11);
29  b11.addActionListener(this);
30  b12=new JButton("*");
31  p.add(b12);
32  b12.addActionListener(this);
33  b13=new JButton("/");
34  p.add(b13);
35  b13.addActionListener(this);
36  b14=new JButton("=");
37  p.add(b14);
38  b14.addActionListener(this);
39  b15=new JButton("C");
40  p.add(b15);
41  b15.addActionListener(this);
42  res=new JTextField(10);
43  add(p, BorderLayout.CENTER);
44  add(res, BorderLayout.NORTH);
45  setVisible(true);
46  setSize(200,200);
47  }
48  public void actionPerformed(ActionEvent ae){
49      JButton pb=(JButton)ae.getSource();
50      if(pb==b15)
51      {
52          r=n1=n2=0;
53          res.setText("");
54      }
55      else if(pb==b14)
56      {
57          n2=Integer.parseInt(res.getText());
58          eval();
59          res.setText(""+r);
60      }
61      else
62      {
63          boolean opf=false;
64          if(pb==b10)
65          { op='+';
66            opf=true;
67          }
68          if(pb==b11)
69          { op='-'; opf=true; }
70          if(pb==b12)
71          { op='*'; opf=true; }
72          if(pb==b13)
73          { op='/'; opf=true; }
74          if(opf==false)
75          {
76              for(i=0;i<10;i++)
77              {
78                  if(pb==b[i])
79                  {
80                      String t=res.getText();
81                      t+=i;
82                      res.setText(t);
83                  }
84              }
85          }
86          else
87          {
88              n1=Integer.parseInt(res.getText());
89              res.setText("");
90          }
91      }
92  }
93  int eval()
94  {
95      switch(op)
96      {
97          case '+': r=n1+n2; break;
98          case '-': r=n1-n2; break;
99          case '*': r=n1*n2; break;
100         case '/': r=n1/n2; break;
101     }
102     return 0;}
103  public static void main(String arg[])
104  {
105      new Calculator();
106  }
107  }
108

```

Output:



Experiment No: 05

Experiment Name: Suppose 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.

Objectives:

1. To understand the concept of race conditions in Java.
2. To learn how to solve race conditions using object and method synchronization.
3. To write a Java program that solves a race condition using object and method synchronization.

Theory:

A race condition is a situation in which two or more threads are accessing the same resource and the outcome of the program depends on the order in which the threads access the resource. This can lead to erroneous and unforeseen results.

There are two ways to solve race conditions in Java: object synchronization and method synchronization.

Object synchronization: Object synchronization ensures that only one thread can access an object at a time. This can be done by using the synchronized keyword on the object's methods.

Method synchronization: Method synchronization ensures that only one thread can execute a particular method at a time. This can be done by using the synchronized keyword on the method's declaration.

Source Code:

```
1 // File Name : Callme.java
2 class Callme {
3     void call(String msg) {
4         System.out.print "[" + msg);
5         try {
6             Thread.sleep(1000);
7         } catch (InterruptedException e) {
8             System.out.println("Interrupted"); }
9         System.out.println("]");
10    } }
11
12 // File Name : Caller.java
13 class Caller implements Runnable {
14     String msg; Callme target; Thread t;
15     public Caller(Callme targ, String s) {
```

```

16         target = targ;
17         msg = s;
18         t = new Thread(this);
19         t.start(); }
20     // synchronize calls to call()
21 public void run() {
22     //synchronized(target) // synchronized block
23     {
24         target.call(msg);
25     }
26 } }
27 //File Name : Synch.java
28 class Synch {
29 public static void main(String args[]) {
30     Callme target = new Callme();
31     Caller ob1 = new Caller(target, "Hello");
32     Caller ob2 = new Caller(target, "Synchronized");
33     Caller ob3 = new Caller(target, "World");
34
35     // wait for threads to end
36     try {
37         ob1.t.join();
38         ob2.t.join();
39         ob3.t.join();
40     } catch (InterruptedException e) {
41         System.out.println("Interrupted");
42     } } }

```

Output:

```

[Hello[Synchronized[World]
]
]

```

Experiment No: 06

Experiment Name: 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.

Objectives:

1. To understand the concept of client-server programming in Java.
2. To learn how to write a TCP socket server in Java.
3. To learn how to write a TCP socket client in Java.
4. To be able to implement a simple chat application using client-server TCP sockets in Java.

Theory:

In client-server programming, there are two entities: the client and the server. The client is the program that initiates the connection, and the server is the program that listens for connection requests.

A TCP socket is a connection-oriented socket that guarantees delivery of data. This means that the data sent by the client will be received by the server, and vice versa.

To implement a client-server TCP socket program in Java, we need to create two classes: the Server class and the Client class.

The Server class will listen for connection requests from clients. When a client connects, the server will create a new thread to handle the connection. The thread will read the message from the client and convert it to uppercase. The thread will then send the message back to the client.

The Client class will connect to the server and send a message to the server. The client will then receive the message back from the server and print it to the console.

Source Code:

Server program

```
1 //SimpleServer.java: A simple //server program
2⊕import java.net.*;
4 public class server1 {
5⊖public static void main(String args[]) throws IOException {
6 // Register service on port 2992
7 ServerSocket s =
8 new ServerSocket(2992);
9 System.out.println("waiting .....");
10 Socket s1=s.accept();
11 //Wait and accept a connection
12 //Get a communication stream //associated
13 //with the socket
14 InputStream s1In = s1.getInputStream();
15 DataInputStream dis = new DataInputStream(s1In);
16 String st = new String (dis.readUTF());
17 System.out.println(st);
18 OutputStream s1out = s1.getOutputStream();
19 DataOutputStream dos =
20 new DataOutputStream (s1.getOutputStream());
21
22 // Send a string!
23 dos.writeUTF(st.toUpperCase());
24
25
26 // Close the connection,
27 //but not the server socket
28 dos.close();
29 s1out.close();
30 s1.close();
31 }
32 }
--
```

Client program

```
1 // SimpleClient.java: A simple client //program
2⊕import java.net.*;
5 public class client1 {
6⊖public static void main(String args[]) throws IOException {
7 //Open your connection to a server, at //port 2992
8 Socket s1 = new Socket("localhost",2992);
9 //Get an input file handle from the //socket and read the input
10 OutputStream s1out = s1.getOutputStream();
11 DataOutputStream dos = new DataOutputStream (s1out);
12 System.out.println("Write Your Message...");// take message from console
13 Scanner sc=new Scanner(System.in);
14 String msg=sc.nextLine();
15 dos.writeUTF(msg); //transfer to server and turn into upper case
16 //dos.writeUTF("hello world");
17 InputStream s1In = s1.getInputStream();
18 DataInputStream dis = new DataInputStream(s1In);
19 String st = new String (dis.readUTF());
20 System.out.println("Output:");
21 System.out.println(st);
22 //When done, just close the connection //and exitdis.close();
23 //s1In.close();
24 //s1.close();
25 //Send a string!
26 //dos.writeUTF("sent from client ");
27 //Close the connection, but not the
28 //server socket
29 //dos.close();
30 //s1out.close();
31 s1.close();
32 }
33 }
```

Output:

```
waiting .....
```

```
Write Your Message...
```

```
Sha Alam
```

```
Output:
```

```
SHA ALAM
```

Experiment No: 07

Experiment Name: 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.

Objectives:

1. To understand the concept of client-server programming in Java.
2. To learn how to write a UDP socket server in Java.
3. To learn how to write a UDP socket client in Java.
4. To be able to implement a simple chat application using client-server UDP sockets in Java.

Theory:

In client-server programming, there are two entities: the client and the server. The client is the program that initiates the connection, and the server is the program that listens for connection requests.

A UDP socket is a connectionless socket that does not guarantee delivery of data. This means that the data sent by the client may not be received by the server, and vice versa.

To implement a client-server UDP socket program in Java, we need to create two classes: the Server class and the Client class.

The Server class will listen for connection requests from clients. When a client connects, the server will send a random delay of 6 ms before reading the message from the client. The server will then convert the message to uppercase and send it back to the client.

The Client class will connect to the server and send a message to the server. The client will then receive the message back from the server and print it to the console.

Source Code:

Server Program:

```
1
2 import java.io.*;
3
4 public class UDPserver
5 {
6     {
7         public static void main(String args[]) throws Exception
8         {
9             DatagramSocket serverSocket = new DatagramSocket(9876);
10             byte[] receiveData = new byte[1024];
11             byte[] sendData = new byte[1024];
12             while(true)
13             {
14                 DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
15                 System.out.println("Waiting..... ");
16                 serverSocket.receive(receivePacket);
17                 System.out.println("Data Received..... ");
18                 String sentence = new String( receivePacket.getData());
19                 System.out.println("RECEIVED: " + sentence);
20                 Thread.sleep(3000);
21                 InetAddress IPAddress = receivePacket.getAddress();
22                 int port = receivePacket.getPort();
23                 // int fa=receivePacket.getLength();
24                 String capitalizedSentence = sentence.toUpperCase();
25                 // int sentencesize = sentence.length();
26                 sendData = capitalizedSentence.getBytes();
27
28                 DatagramPacket sendPacket = new DatagramPacket(sendData,sendData.length ,IPAddress, port);
29                 serverSocket.send(sendPacket);
30             }
31         }
32     }
```

Client Program:

```
1 import java.io.*;
2 import java.net.*;
3
4 public class UDPClient
5 {
6     public static void main(String args[]) throws Exception
7     {
8         System.out.println("write something ");
9         BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
10        DatagramSocket clientSocket = new DatagramSocket();
11        InetAddress IPAddress = InetAddress.getByName("localhost");
12
13        byte[] sendData = new byte[1024];
14        byte[] receiveData = new byte[1024];
15        String sentence = inFromUser.readLine();
16
17
18        sendData = sentence.getBytes();
19
20        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
21        clientSocket.send(sendPacket);
22        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
23        System.out.println("Waiting..... ");
24        clientSocket.receive(receivePacket);
25        System.out.println("Data Received..... ");
26        String modifiedSentence = new String(receivePacket.getData());
27        System.out.println("FROM SERVER:" + modifiedSentence);
28        clientSocket.close();
29    }
30 }
```

Output:

```
write something
Sha Alam
Waiting.....
Data Received.....
FROM SERVER:SHA ALAM
```

Experiment No: 08

Experiment Name: 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 questions:

- Write down a java program to insert data into the Student table of the ICE_PUST database.
- Create a java program to print all student records from the Student table of the ICE_PUST database

Objectives:

- To learn how to connect to an MS Access database using JDBC.
- To learn how to insert data into an MS Access database using JDBC.
- To learn how to print data from an MS Access database using JDBC.

Theory:

JDBC (Java Database Connectivity) is a Java API that allows Java programs to connect to and manipulate databases. To connect to an MS Access database using JDBC, we need to use the DriverManager class. The DriverManager class provides a method called getConnection() that can be used to get a connection to a database.

To insert data into an MS Access database using JDBC, we need to use the Statement class. The Statement class provides a method called executeUpdate() that can be used to execute SQL statements. The SQL statement that we need to use to insert data into the database is:

SQL

```
INSERT INTO Student (Name, Email, Phone) VALUES ('John Doe', 'johndoe@example.com', '123-456-7890');
```

To print data from an MS Access database using JDBC, we can use the ResultSet class. The ResultSet class represents the results of a query. We can use the next() method of the ResultSet class to iterate through the results and print the data.

Source Code:

Connection program:

```
1 import java.sql.*;
2 public class java_sql_db_connect {
3     public static void main(String[] args){
4         try{
5             Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");//Loading Driver
6             Connection connection= DriverManager.getConnection("jdbc:ucanaccess://E:\\Course\\3-2\\
7                 + "Network Programming with Java\\Lab\\Solution\\Database\\Lab_18.accdb");//Establishing Connection
8             System.out.println("Connected Successfully");
9         }
10        }catch(Exception e){
11            System.out.println("Error in connection"+e);
12        }
13    }
14 }
15 }
16 }
```

Show Program:

```
1 import java.sql.*;
2 public class java_sql_db_show_data {
3     public static void main(String[] args){
4         try{
5             Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");//Loading Driver
6             Connection connection= DriverManager.getConnection("jdbc:ucanaccess://E:\\Course\\3-2\\Network Programming with
7                 System.out.println("Connected Successfully");
8             //Using SQL SELECT Query
9             PreparedStatement preparedStatement=connection.prepareStatement("select * from Practice");
10            //Creating Java ResultSet object
11            ResultSet resultSet=preparedStatement.executeQuery();
12            System.out.println(resultSet);
13            System.out.println("Name      "+"Email      "+"Phone");
14            while(resultSet.next()){
15                String name=resultSet.getString("Name");
16                String email=resultSet.getString("Email");
17                String phone=resultSet.getString("Phone");
18                //Printing Results
19                System.out.println(name+" "+"email+" "+"phone");
20            }
21        }
22        }catch(Exception e){
23            System.out.println("Error in connection");
24        }
25    }
26 }
27 }
28 }
```

Output:

```
Connected Successfully
net.ucanaccess.jdbc.UcanaccessResultSet@48f2bd5b
Name      Email      Phone
Md Sha Alam shaalam.ice@pust.ac.bd 01518682123
Rubel Hossain rubel.ice@pust.ac.bd 01722633140
Sofiq sofiq@gmail.com 01557240628
Ahmed rabbi.ru@gmail.com 01722633140
```

Experiment No: 09

Experiment Name: 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 the following questions:

- Write down a java program to create a GUI registration form according to the Teacher table.
- Create a java program to insert data into the Teacher table of the Lab_18 database from the GUI registration form which you have already created.

Objectives:

- To learn how to create a GUI registration form using Java Swing.
- To learn how to insert data into an MS Access database using JDBC.

Theory:

Java Swing is a GUI toolkit that is used to create graphical user interfaces in Java. It provides a variety of components that can be used to create forms, dialog boxes, menus, and other GUI elements.

JDBC (Java Database Connectivity) is a Java API that allows Java programs to connect to and manipulate databases. It provides a variety of classes and interfaces that can be used to perform database operations, such as connecting to a database, creating a statement, executing a query, and retrieving results.

Source Code:

Form Program:

```
1 import java.awt.*;
2
3 public class java_sql_gui_1_simple extends JFrame implements ActionListener
4 {
5     JTextField tf1,tf2,tf3;
6     JLabel l1,l2,l3;
7     JButton b1,b2,b3,b4;
8     Connection con;
9     ResultSet rs;
10    Statement st;
11    public java_sql_gui_1_simple()
12    {
13        super("personDetails");
14        setLayout(null);
15        Label d=new Label("Person Details");
16        d.setBounds(200,10,150,50);
17        d.setBackground(Color.RED);
18        add(d);
19        l1=new JLabel("Name");
20        l1.setBounds(100,100,100,50);
21        add(l1);
22        tf1=new JTextField(" ");
23        tf1.setBounds(200,100,250,40);
24        add(tf1);
25        l2=new JLabel("Email");
26        l2.setBounds(100,150,100,50);
27        add(l2);
28        tf2=new JTextField();
29        tf2.setBounds(200,150,250,40);
30        add(tf2);
31        l3=new JLabel("Phone");
32        l3.setBounds(100,200,100,50);
33        //l3.setFont(new Font("",Font.BOLD));
34        add(l3);
35        tf3=new JTextField();
36        tf3.setBounds(200,200,250,40);
37        add(tf3);
38        b1=new JButton("Show");
39        b1.setBounds(100,250,90,50);
40        add(b1);
41        b2=new JButton("Insert");
42        b2.setBounds(200,250,90,50);
43        add(b2);
44        b3=new JButton("Update");
45        b3.setBounds(300,250,90,50);
46        add(b3);
47        b4=new JButton("Delete");
48        b4.setBounds(400,250,90,50);
49        add(b4);
50        b1.addActionListener(this);
51        b2.addActionListener(this);
```

```

57     b3.addActionListener(this);
58     b4.addActionListener(this);
59     setSize(600,600);
60     setVisible(true);
61     connection();
62 }
63
64 public void connection()
65 {
66     try
67     {
68         Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");//Loading Driver
69         con= DriverManager.getConnection("jdbc:ucanaccess://D:\\Database\\Lab_18.accdb");//Establishing Connection
70         System.out.println("Connected Successfully");
71         st=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);
72         rs=st.executeQuery("SELECT * FROM Practice");
73         System.out.println("connect");
74
75         //JOptionPane.showMessageDialog(null, "please click next Button");
76     }catch(Exception e)
77     {
78         System.out.println(e);
79     }
80 }
81
82 public void actionPerformed(ActionEvent e)
83 {
84
85     if(e.getSource()==b1) {
86         System.out.println("I am show");
87         try
88         {
89             if(rs.next()){
90                 tf1.setText(rs.getString("Name"));
91                 tf1.setText(rs.getString("Name"));
92                 tf2.setText(rs.getString("Email"));
93                 tf3.setText(rs.getString("Phone"));
94             }
95             else
96             {
97                 JOptionPane.showMessageDialog(null, "No data");
98             }
99         }catch(Exception ex)
100        {
101
102        }
103    }
104    if(e.getSource()==b2)
105    {
106        System.out.println("I am Insert");
107        String up1=tf1.getText();
108        String up2=tf2.getText();
109        String up3=tf3.getText();
110        try
111        {
112            rs.updateString("Name",up1);
113            rs.updateString("Email",up2);
114            rs.updateString("Phone",up3);
115            rs.insertRow();
116            System.out.println("one row is inserted");
117        }
118        catch(Exception er)
119        {
120
121        }
122    }
123    if(e.getSource()==b3)
124    {
125        System.out.println("I am Update");
126        String up1=tf1.getText();
127        String up2=tf2.getText();
128        String up3=tf3.getText();
129        try
130        {
131            rs.updateString("Name",up1);
132            rs.updateString("Email",up2);
133            rs.updateString("Phone",up3);
134            rs.updateRow();
135            System.out.println("one row is updated");
136        }
137        catch(Exception er)
138        {
139
140        }
141    }
142    if(e.getSource()==b4)
143    {
144        System.out.println("I am Delete");
145        try
146        {
147            rs.deleteRow();
148            System.out.println("one row is deleted");
149        }catch(Exception er)
150        {
151
152        }
153    }
154 }
155 public static void main(String a[])
156 {
157     new java_sql_gui_1_simple();
158 }

```

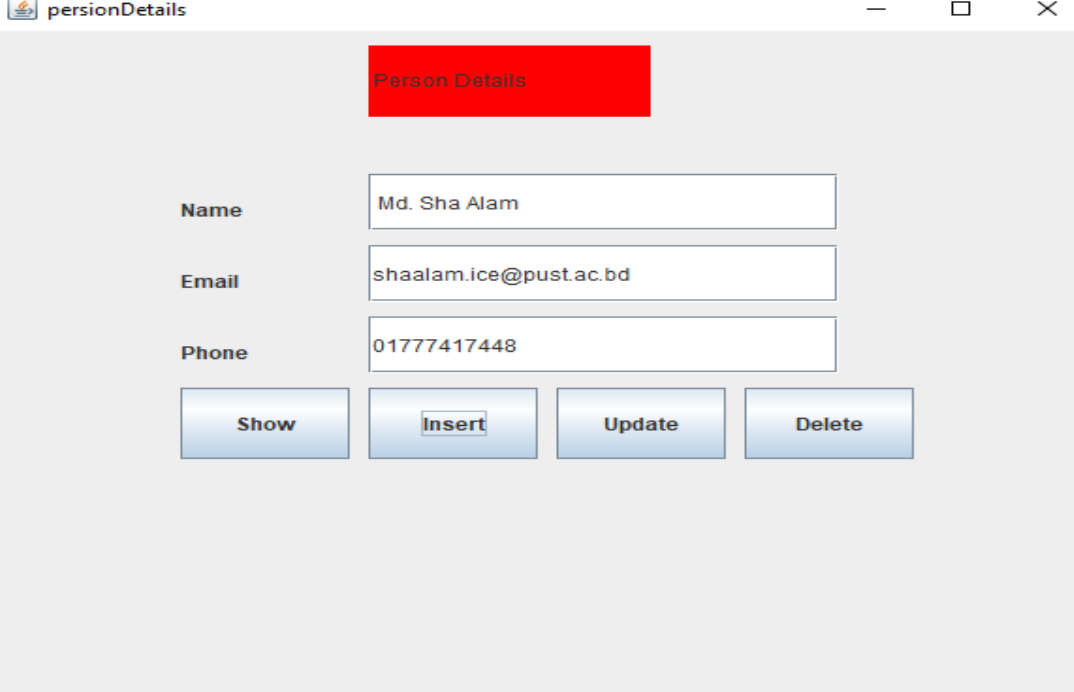
Update Program:

```
import java.sql.Connection;
public class java_sql_db_update_data {
    public static void main(String[] args){
        try{
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");//Loading Driver
            Connection connection= DriverManager.getConnection("jdbc:ucanaccess://E:\\Course\\3-2\\Network Programming with
            System.out.println("Connected Successfully");
            //Crating PreparedStatement object
            PreparedStatement preparedStatement=connection.prepareStatement("update Practice set Name=? where Phone=?");
            //Setting values for Each Parameter
            preparedStatement.setString(1,"Ahmed");
            preparedStatement.setString(2,"01722633140");

            preparedStatement.executeUpdate();
            System.out.println("data updated successfully");

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

Output:



personDetails

Person Details

Name: Md. Sha Alam

Email: shaalam.ice@pust.ac.bd

Phone: 01777417448

Show Insert Update Delete