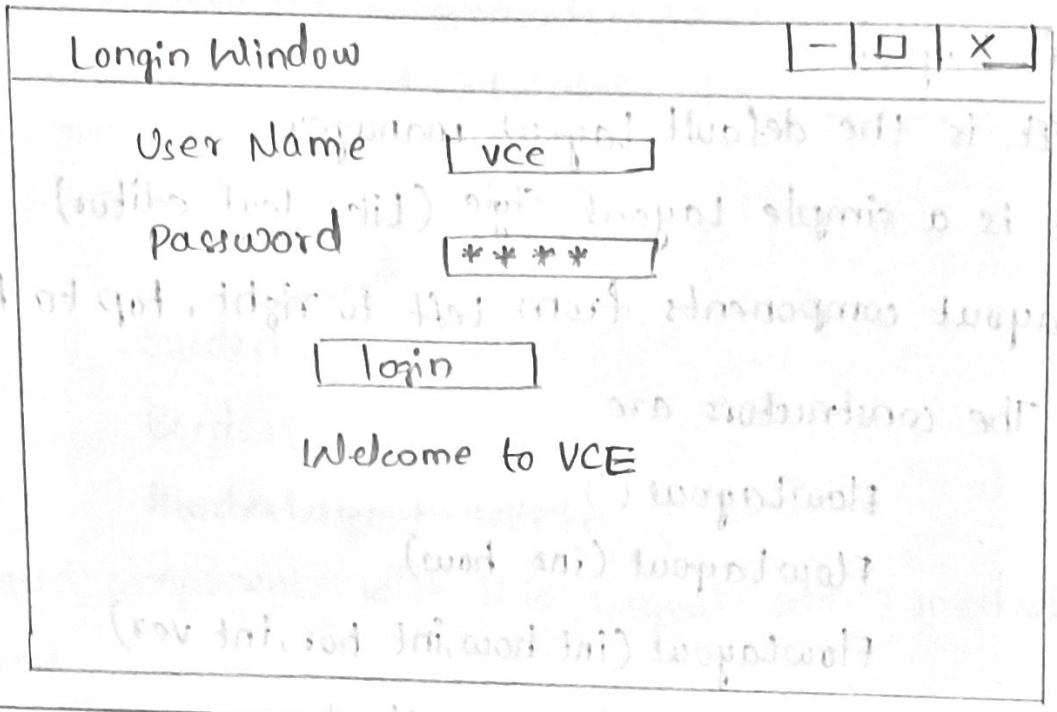
1. Write a Java program to Illustrate AWT controls. Design Login Form using AWT controls.

```
import java.awt.*;
import java.awt.event.*;
public class LoginDemo extends Frame implements ActionListener
{
    Label l1, l2, l3;
    TextField t1, t2;
    Button b1;
    LoginDemo(String name)
    {
        super(name);
        l1 = new Label("User Name");
        l1.setBounds(20,50,100,20);
        l2 = new Label("Password");
        l2.setBounds(20,100,100,20);
        t1 = new TextField();
        t1.setBounds(130,50,100,20);
        t2 = new TextField();
        t2.setBounds(130,100,100,20);
        t2.setBounds()
        t2.setEchoChar('*');
        b1 = new Button("login");
        b1.setBounds(80,150,80,20);
```

```
        l3 = new Label("→");
        l3.setBounds(20,200,200,20);
        add(l1); add(t1);
        add(l2); add(t2);
        add(b1); add(l3);
        b1.addActionListener(this);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
            dispose();
        }});
}
public void actionPerformed(ActionEvent e)
    {
        String uname = t1.getText();
        String pwd = t2.getText();
        if(uname.equals("vce") && pwd.equals("root"))
                l3.setText("Welcome to VCE");
        else
            l3.setText("Invalid Username or Password");
    }
    public static void main(String[] args)
    {
        new LoginDemo("Login Window");
    }
}
```

output :-



2. Write a note on Layout Managers.

→ A Layout Manager automatically arranges user controls within a window by using an algorithm.

⇒ Positioning manually the controls on a large window is more difficult.

⇒ Java provides Layout managers to position the controls.

→ Container object has a layout manager associated with it.

→ A layout manager is an instance of any class that implements "LayoutManager" interface.

→ A Layout manager is set by using

void setLayout (LayoutManager obj)

to set manually then obj is null.

→ Java has several predefined LayoutManager classes.

1) Flow Layout

→ It is the default layout manager.

→ It is a simple layout style (Like text editor).

→ Layout components from left to right, top to bottom.

The constructors are

FlowLayout()
FlowLayout(int how)
FlowLayout(int how, int hor, int ver)

How each line is aligned is specified "how".

The values are

FlowLayout.LEFT
FlowLayout.CENTER
FlowLayout.RIGHT
FlowLayout.LEADING
FlowLayout.TRAILING

2) Border Layout

→ It is used for top-level windows

→ The layout has four narrow, fixed-width component at the edges and one large area in the center.

→ The four sides are called north, south, east & west.

The constructors are

BorderLayout()
BorderLayout(int hor, int ver)

hor, ver are used to specify the horizontal and vertical space left between components.

The constructor constants defined in BorderLayout are

BorderLayout.CENTER

BorderLayout.EAST

BorderLayout.WEST

BorderLayout.SOUTH

BorderLayout.NORTH

To add components with this Layout add() method is used

void add(Component obj, Object region)

3) Grid Layout

⇒ Grid Layout lays out components in a two-dimensional grid.

⇒ The no. of rows & columns are specified while creating an object

→ The Constructors are

GridLayout() - Creates a single column grid layout

GridLayout(int rows, int cols) -

creates a grid layout with the specified no. of rows & cols.

GridLayout(int rows, int cols, int hor, int ver)

-creates a grid layout with the specified no. of rows & cols, the spacing between

components is specified using hor & ver.

rows = 0 ⇒ allows unlimited length col

col = 0 ⇒ allows unlimited length rows.

4) Card Layout

⇒ This Layout manager stores several different layouts.

The constructors are

CardLayout() – creates a default card layout.

CardLayout(int hor, int ver) –

– Allows to specify the horizontal & vertical space left b/w components.

⇒ To add panels to the main card (panel) add() method is used.

void add(Component panel, Object name)

⇒ The method defined by the CardLayout are

void first(Container deck)

void left(Container deck)

void next(Container deck)

void previous(Container deck)

void show(Container deck, String card name)

show() method displays the card whose name is passed as name.

3. How is Multithreaded Programming in Java.

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.

Each part of such program is called a thread. Threads can be created by using two mechanisms:

1. Extending the Thread class
2. Implementing the Runnable Interface

1) Thread Creation by extending the Thread class

We create a class that extends the java.lang.Thread class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

code

```
class MultithreadingDemo extends Thread
{
    public void run()
    {
        try {
            System.out.println("Thread" + Thread.
                currentThread().getId()
                + " is running");
    }
```

```
    catch (Exception e)
        {
            System.out.println("Exception is caught");
        }
    }
}
public class Multithread
{
    public static void main (String[] args)
    {
        int n = 8;
        for (int i=0; i<n; i++)
        {
            MultithreadingDemo object
                = new MultithreadingDemo();
            object.start();
        }
    }
}
```

output :-

```
Thread 15 is running
"    14    "
"    16    "
"    12    "
"    11    "
"    13    "
"    18    "
"    17    "
```

2) Thread creation by implementing the Runnable interface

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

code

```
class MultithreadingDemo implements Runnable
{ public void run()
  { try {
      System.out.println ("Thread " + Thread.current
                          Thread().getId() +
                          " is running");
    }
    catch (Exception e)
    { System.out.println ("Exception is caught");
    }
  }
}
class Multithread
{ public static void main (String[] args)
  { int n= 8;
    for (int i=0; i<n; i++)
    { Thread object = new Thread (new Multithreading
                                  Demo());
      object.start();
    }
  }
}
```

output :-

Thread 13 is running
" 11 "
" 12 "
" 15 "
" 14 "
" 18 "
" 17 "
" 16 "

Programs

1. Write a program to demonstrate Exception Handling using single try and multiple catch blocks.

```
import java.util.*;
public class ExcepDemo1
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number num1");
        String s1 = sc.next();
        System.out.println("Enter the number num2");
        String s2 = sc.next();
        try
        {
            int num1 = Integer.parseInt(s1);
            int num2 = Integer.parseInt(s2);
```

```
        System.out.println ("num1 is" +num1);
        System.out.println ("num2 is" +num2);
        if(num2==0)
            throw new ArithmeticException ("Division Error");
        int res = num1/num2;
            System.out.println ("The result is" +res);
    }
    catch (NumberFormatException e)
    {
        System.out.println ("The numbers must be numeric data");
        System.out.println ("Exception" +e);
    }
    catch (ArithmeticException e)
    {
        System.out.println ("num2 must not be zero");
        System.out.println ("Exception" +e);
    }
    finally
    {
        System.out.println ("Finally block is executed");
    }
    System.out.println("Remaining statements");
    }
}
```

output:-

```
Enter the number nums
12
Enter the number num2
3
num1 is 12
num2 is 3
The result is 4
Finally block is executed
```

2. Write a program to demonstrate Multithreading.

```
class NewThread extends Thread
{
    NewThread (String name)
    {
        super(name);
    }
    public void run()
    {
        try {
            for (int i=1 ; i<=4 ; i++)
            {
                System.out.println("Java is object
                                    oriented" + getName());
                sleep(1000);
            }
        }
        catch (InterruptedException ie)
        {
            System.out.println ("Child Thread - Exception
                                 caught");
        }
    }
}
public class ThreadDemo
{
    public static void main (String args[])
    {
        NewThread t1 = new NewThread ("First");
        NewThread t2 = new NewThread ("Second");
            t1.start();
            t2.start();
        System.out.println("Main program");
    }
}
```