

Practical No. 1

Aim: -

Write a program to demonstrate the use of AWT components like Label, Text field, Text Area, Button, Checkbox, Radio Button etc.

Theory:-

Frame: -

Frame is a Window class child and comprises the title bar, border and menu bars. Therefore, the frame provides a resizable canvas and is the most widely used container used for developing AWT-based applications. Various components such as buttons, text fields, scrollbars etc., can be accommodated inside the frame container.

Panel: -

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Button: -

This is used to create a button on the user interface with a specified label. We can design code to execute some logic on the click event of a button using listeners.

Text Fields: -

This component of java AWT creates a text box of a single line to enter text data.

Label: -

This component of java AWT creates a multi-line descriptive string that is shown on the graphical user interface.

Checkbox: -

This component is used to create a checkbox of GUI whose state can be either checked or unchecked.

Text Area: -

The Text Area control in AWT provides us multiline editor area. The user can type here as much as they want. When the text in the text area become larger than the viewable area the scroll bar is automatically appears which help us to scroll the text up & down and right & left.

Code: -

```
import java.awt.*;
//import java.awt.event.*;
public class Pract_1 extends Frame {
    Pract_1(){
        Label title,name,gender,adhar,address;
        CheckboxGroup c;
        TextField tname;
        Checkbox male,female,yes,no;
        TextArea t;
        Button submit,reset;

        Panel p = new Panel();
        add(p);
        p.setBounds(40,20,400,600);
        p.setBackground(Color.GRAY);
        p.setLayout(null);

        title = new Label("STUDENT INFORMATION");
        p.add(title);
        title.setBounds(120,30,150,25);
        title.setBackground(Color.CYAN);

        name = new Label("Name");
        p.add(name);
        name.setBounds(40,100,50,30);
        //name.setBackground(Color.BLUE);

        //textbox for name
        tname = new TextField();
        p.add(tname);
        tname.setBounds(90,100,150,25);

        //gender
```

```
p.add( gender = new Label("Gender"));
gender.setBounds(40,135,50,25);

//gender male feamle
c = new CheckboxGroup();

p.add(male = new Checkbox("Male", false, c));
male.setBounds(90,160,50,25);

p.add(female = new Checkbox("Female",false , c));
female.setBounds(150,160,70,25);

//adhar is available or not
p.add(adhar = new Label("Aadhar"));
adhar.setBounds(40,195,70,25);

p.add(yes = new Checkbox("Yes", false));
yes.setBounds(90,220,50,25);
p.add(no = new Checkbox("No",false));
no.setBounds(150,220,70,25);

//Address
p.add(address = new Label("Enter Address"));
address.setBounds(40,250,80,25);

p.add(t = new TextArea(40,40));
t.setBounds(70,280,200,70);

//Creating Button
p.add(reset= new Button("RESET"));
reset.setBounds(120,380,50,25);
reset.setBackground(Color.BLACK);
reset.setForeground(Color.WHITE);

p.add(submit= new Button("SUBMIT"));
submit.setBounds(200,380,50,25);
```

```
        submit.setBackground(Color.BLACK);
        submit.setForeground(Color.WHITE);

        setTitle("Student Information");
        setLayout(null);
        setSize(500,800);
        setVisible(true);
    }
    public static void main(String args[]) {
        new Pract_1();
    }
}
```

Output:-

A screenshot of a Java Swing window titled "Student Information". The window has a standard title bar with minimize, maximize, and close buttons. The content area has a dark gray background. At the top, there is a cyan rectangular label with the text "STUDENT INFORMATION". Below this, the form contains the following fields and controls:

- Name:** A text field containing the text "Gaurav Naraware".
- Gender:** Two radio buttons labeled "Male" (which is selected) and "Female".
- Aadhar:** Two checkboxes labeled "Yes" (which is checked) and "No".
- Enter Address:** A text area containing the email address "gauravnaraware3112003@gmail.com".
- Buttons:** Two buttons labeled "RESET" and "SUBMIT" are positioned at the bottom of the form.

A screenshot of a Java Swing window titled "Student Information". The window has a standard title bar with minimize, maximize, and close buttons. The content area has a dark gray background. At the top, there is a cyan rectangular label with the text "STUDENT INFORMATION". Below this, the form contains the following fields and controls:

- Name:** An empty text field.
- Gender:** Two radio buttons labeled "Male" and "Female", both of which are unselected.
- Aadhar:** Two checkboxes labeled "Yes" and "No", both of which are unchecked.
- Enter Address:** An empty text area.
- Buttons:** Two buttons labeled "RESET" and "SUBMIT" are positioned at the bottom of the form.

Practical No. 2

Aim: -

Write a program to design a form using the components List and Choice.

Theory:-

Java AWT Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

AWT Choice Class Declaration

1. **public class** Choice **extends** Component **implements** ItemSelectable, Accessible

Choice Class constructor

Sr. no.	Constructor	Description
1.	Choice()	It constructs a new choice menu.

Java AWT List

The object of List class represents a list of text items. With the help of the List class, user can choose either one item or multiple items. It inherits the Component class.

AWT List class Declaration

1. **public class** List **extends** Component **implements** ItemSelectable, Accessible

AWT List Class Constructors

Sr. no.	Constructor	Description
---------	-------------	-------------

1.	List()	It constructs a new scrolling list.
2.	List(int row_num)	It constructs a new scrolling list initialized with the given number of rows visible.
3	List(int row_num, Boolean multipleMode)	It constructs a new scrolling list initialized which displays the given number of rows.

Java AWT Label

The **object** of the Label class is a component for placing text in a container. It is used to display a single line of **read only text**. The text can be changed by a programmer but a user cannot edit it directly.

It is called a passive control as it does not create any event when it is accessed. To create a label, we need to create the object of **Label** class.

AWT Label Class Declaration

1. **public class** Label **extends** Component **implements** Accessible

Sr. no.	Constructor	Description
1.	Label()	It constructs an empty label.
2.	Label(String text)	It constructs a label with the given string (left justified by default).

3	Label(String text, int alignment)	It constructs a label with the specified string and the specified alignment.
---	-----------------------------------	--

Java AWT TextField

The [object](#) of a **TextField** class is a text component that allows a user to enter a single line text and edit it. It inherits **TextComponent** class, which further inherits **Component** class.

When we enter a key in the text field (like key pressed, key released or key typed), the event is sent to **TextField**. Then the **KeyEvent** is passed to the registered **KeyListener**. It can also be done using **ActionEvent**; if the **ActionEvent** is enabled on the text field, then the **ActionEvent** may be fired by pressing return key. The event is handled by the **ActionListener** interface.

Java AWT Button

A button is basically a control component with a label that generates an event when pushed. The **Button** class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

When we press a button and release it, AWT sends an instance of **ActionEvent** to that button by calling **processEvent** on the button. The **processEvent** method of the button receives the all the events, then it passes an action event by calling its own method **processActionEvent**. This method passes the action event on to action listeners that are interested in the action events generated by the button.

Code: -

```
import java.awt.*;

public class Pract_2 extends Frame {
    Pract_2(){
        Label name ,batch,sub;
        TextField tname;
        Choice C_batch;
        List L_sub;
        Button Submit;
        Panel p = new Panel();
        add(p);
        p.setLayout(null);
        p.setBounds(25,50,400,450);
        p.setBackground(Color.LIGHT_GRAY);

        //name of the student
        p.add(name = new Label("Name: "));
        name.setBounds(40,100,50,30);

        p.add(tname = new TextField());
        tname.setBounds(140,100,150,25);

        // creating a batch Label
        p.add(batch = new Label("Select Batch:"));
        batch.setBounds(40,140,100,25);
        // batch.setBackground(Color.BLUE);

        // creating a choice for batch
        p.add(C_batch = new Choice());
        C_batch.add("Batch CO-1");
        C_batch.add("Batch CO-2");
        C_batch.add("Batch CO-3");
        C_batch.setBounds(140,140,100,25);
```

```
// creating a list
p.add(sub = new Label(" Select Practical: "));
sub.setBounds(40,220,100,25);

p.add( L_sub = new List(1,false));
L_sub.add("Advanced java");
L_sub.add("Operating System");
L_sub.add("Client side scripting");
L_sub.add("Software Testing");
L_sub.add("Environmental Studies");
L_sub.setBounds(140,220,200,100);

//Creating a submit Button
p.add(Submit = new Button("Submit"));
Submit.setBounds(160,370,100,25);
Submit.setBackground(Color.BLACK);
Submit.setForeground(Color.WHITE);

setLayout(null);
setTitle("Practical no 2");
setVisible(true);
setSize(450,550);

}

public static void main(String args[]) {
    new Pract_2();
}
}
```

Output: -

Practical no 2

Name:

Select Batch:

Select Practical:

Advanced java
Operating System
Client side scripting
Software Testing
Environmental Studies

Practical no 2

Name:

Select Batch:

Batch CO-1
Batch CO-1
Batch CO-2
Batch CO-3

Select Practical:

Advanced java
Operating System
Client side scripting
Software Testing
Environmental Studies

Practical No. 3

Aim:-

Write a program to design a Simple Calculator with the use of Grid Layout.

Theory:-

Java GridLayout

GridLayout class represents a layout manager with a specified number of rows and columns in a rectangular grid. The GridLayout container is divided into an equal-sized of rectangles, and one of the components is placed in each rectangle. Every rectangle cell has the same size therefore, they contain a component, which fills the entire cell. When the user changes or adjusts the size of the container, the size of each rectangles changes accordingly.

Constructors of the class:

1. **GridLayout():** It Creates a grid layout with a default of one column per component, in a single row.
2. **GridLayout(int rw, int cl):** It creates a grid layout with the specified number of rows and columns.
3. **GridLayout(int rw, int cl, int hgap, int vgap):** It creates a grid layout with the specified number of rows and columns with horizontal and vertical gap.

Commonly Used Methods:

- **addLayoutComponent(String str, Component cmp):** Adds the specified component with the specified name to the layout.
- **setColumns(int cl):** Sets the number of columns in this layout to the specified value.
- **setHgap(int hgap):** Sets the horizontal gap between components to the specified value.
- **setRows(int rw):** Sets the number of rows in this layout to the specified value.
- **setVgap(int vgap):** Sets the vertical gap between components to the specified value.

- **layoutContainer(Container pr):** Lays out the specified container using this layout.
- **toString():** Returns the string representation of this grid layout's values

Code :-

```
import java.awt.*;

public class Pract_3 extends Frame {

    Pract_3() {
        Panel f = new Panel();
        add(f);
        f.setBackground(Color.BLUE);

        Label calcs = new Label("CALCULATOR");
        add(calcs);
        calcs.setBounds(110,40,85,25);
        calcs.setBackground(Color.CYAN);

        Button b, b1;
        f.add(b1= new Button("AC"));
        f.add(b1= new Button("X"));
        f.add(b1= new Button("%"));
        f.add(b1= new Button("/"));

        for (int i = 9; i > 6; i--) {
            String a = Integer.toString(i);
            b = new Button(a);
            f.add(b);
        }
        f.add(b=new Button("*"));
        for (int i = 6; i > 3; i--) {
            String a = Integer.toString(i);
            b = new Button(a);
```

```
        f.add(b);
    }
    f.add(b=new Button("-"));
    for (int i = 3; i > 0; i--) {
        String a = Integer.toString(i);
        b = new Button(a);
        f.add(b);
    }
    f.add(b=new Button("+"));
    f.add(b=new Button("."));
    f.add(b=new Button("0"));
    f.add(b=new Button("^"));

    b1=new Button("=");
    f.add(b1);
    // f.add(b=new Button(""))

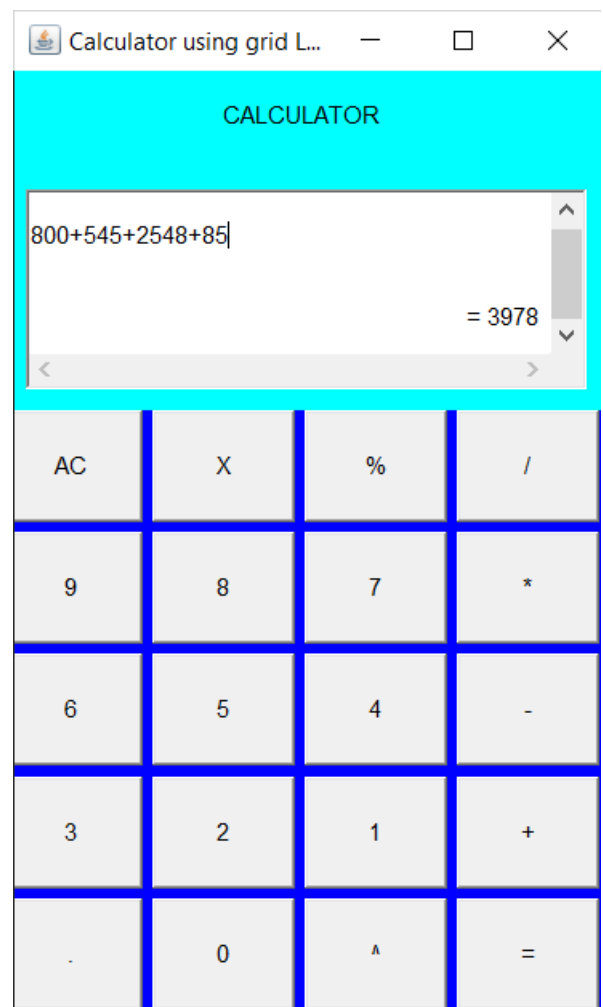
    // b1.setBounds(85,500,80,100);

    f.setLayout(new GridLayout(5, 4, 5, 5));
    setLayout(null);
    f.setBounds(0, 200, 300, 300);
    setSize(310, 510);
    f.setVisible(true);
    setVisible(true);
    setBackground(Color.CYAN);
    setTitle("Calculator using grid Layout");

    TextArea te = new TextArea("");
    te.setBounds(13,90,280,100);
    add(te);
}
```

```
public static void main(String args[]) {  
    new Pract_3();  
}  
}
```

Output :-



Practical No. 4

Aim :-

Write a Program to create a two level card deck that allow user to select component of panel using a card Layout.

Theory :-

CardLayout

The **Java CardLayout** class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout Class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly Used Methods of CardLayout Class

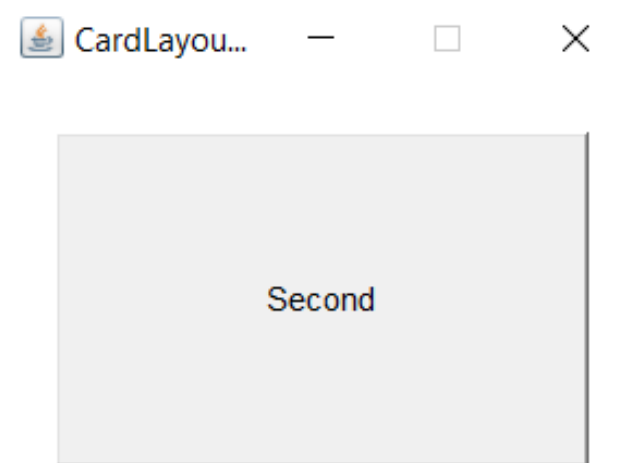
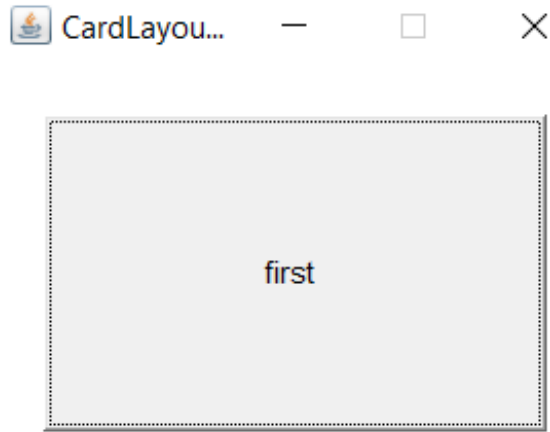
- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name

Code:-

```
import java.awt.*;
import java.awt.event.*;
class CardLayoutExample extends Frame implements ActionListener
{
    CardLayout card = new CardLayout(20,20);
    CardLayoutExample()
    {
        setLayout(card);
        Button Btnfirst = new Button("first ");
        Button BtnSecond = new Button ("Second");
        Button BtnThird = new Button("Third");
        add(Btnfirst,"Card1");
        add(BtnSecond,"Card2");
        add(BtnThird,"Card3");
        Btnfirst.addActionListener(this);
        BtnSecond.addActionListener (this);
        BtnThird.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
        card.next(this);
    }
}

class CardLayoutJavaExample
{
    public static void main(String args[])
    {
        CardLayoutExample frame = new CardLayoutExample();
        frame.setTitle("CardLayout in Java Example");
        frame.setSize(250,200);
        frame.setResizable(false);
        frame.setVisible(true);
    }
}
```

Output :-



Practical No. 5

Aim :-

Write a Program using AWT to create a menu bar where menu bar contains menu items such as File, Edit, View and create a submenu under the File menu: New and Open.

Theory :-

Menu, MenuItem and MenuBar class

In this article, we are going to understand how to add a menu bar, menu and its menu items to the window application.

- A menu bar can be created using **MenuBar** class.
- A menu bar may contain one or multiple menus, and these menus are created using **Menu** class.
- A menu may contain one of multiple menu items and these menu items are created using **MenuItem** class.

Simple constructors of MenuBar, Menu and MenuItem

Constructor	Description
public MenuBar()	Creates a menu bar to which one or many menus are added.
public Menu(String title)	Creates a menu with a title.
public MenuItem(String title)	Creates a menu item with a title.

Code :-

```
import java.awt.*;
// import java.awt.event.*;

public class Pract_5 extends Frame{
    Pract_5(){
        MenuBar m = new MenuBar();
        setMenuBar(m);
        setTitle("Practical 5");
        setSize(600,400);
        setVisible(true);
        // setDefaultCloseOperation(EXIT_ON_CLOSE);
        Menu m1 = new Menu("File");
        Menu m2 = new Menu("Edit");
        Menu m3 = new Menu("View");
        Menu m4 = new Menu("Exit");
        m.add(m1);
        m.add(m2);
        m.add(m3);
        m.add(m4);

        MenuItem i1 = new MenuItem("New");
        MenuItem i2 = new MenuItem("Open");

        m1.add(i1);
        m1.addSeparator();
        m1.add(i2);
    }

    public static void main (String args[]){
        new Pract_5();
    }
}
```

Output :-



Practical No. 6

Aim :-

Write a program using swing to display a scrollpane and JComboBox in an JApplet with the items -English, marathi, hindi, Sanskrit.

Theory:-

Java JScrollPane

A JScrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

The layout manager used by JScrollPane. *JScrollPaneLayout* is based on nine components: a viewport, two scrollbars, a row header, a column header, and four “corner” components.

Constructor of the class:

- **ScrollPaneLayout():** It is used to Construct a new ScrollPaneLayout.

Commonly Used Methods:

1. **removeLayoutComponent(Component comp):** Removes the specified component from the layout.
2. **getColumnHeader():** It returns the JViewport object that is the column header.
3. **getVerticalScrollBar():** Returns the JScrollBar object that handles the vertical scrolling.
4. **getHorizontalScrollBar():** Returns the JScrollBar object that handles the horizontal scrolling.
5. **addLayoutComponent(String st, Component c):** Adds the specified component to the layout.
6. **getViewport():** Returns the JViewport object that displays the scrollable contents.
7. **getCorner(String key):** It is used to returns the Component at the specified corner.

Code:

```
import javax.swing.*;
import java.awt.*;

public class JScrollPaneDemo
{
    public static void main(String[] args)
    {
        /* Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
        */

        JFrame JFrameMain = new JFrame();
        JFrameMain.setVisible(true);
        JFrameMain.setSize(400,400);

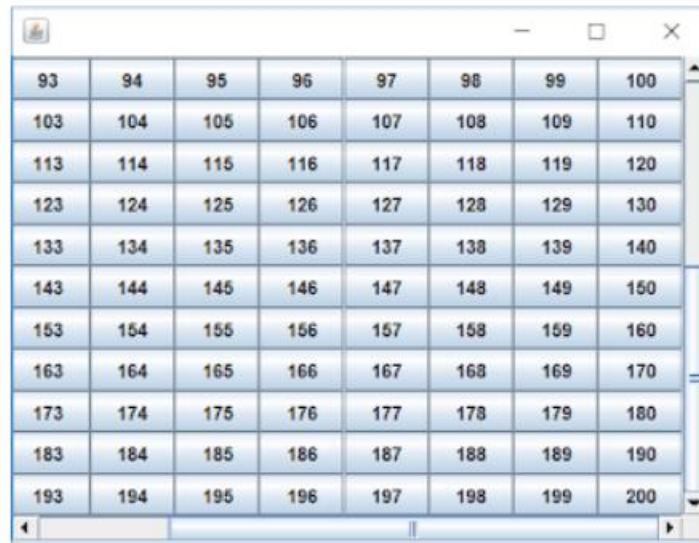
        JPanel JPanelButton = new JPanel();
        JPanelButton.setLayout(new GridLayout(20,10));
        for( int i = 1 ; i <= 200 ; i++ )
        {
            String s = "";
            s = s.valueOf(i);
            JPanelButton.add(new JButton( s ) );
        }

        int v = JScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
        int h = JScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED ;

        JScrollPane JScrollPaneObj = new JScrollPane(JPanelButton , v,h);

        JFrameMain.add(JScrollPaneObj , BorderLayout.CENTER);
    }
}
```


Output:



93	94	95	96	97	98	99	100
103	104	105	106	107	108	109	110
113	114	115	116	117	118	119	120
123	124	125	126	127	128	129	130
133	134	135	136	137	138	139	140
143	144	145	146	147	148	149	150
153	154	155	156	157	158	159	160
163	164	165	166	167	168	169	170
173	174	175	176	177	178	179	180
183	184	185	186	187	188	189	190
193	194	195	196	197	198	199	200

Java JComboBox:

Code:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

public class JComboBoxDemo extends JApplet implements ItemListener
{
    JLabel JLabelObj ;

    public void init()
    {
        setLayout(new FlowLayout());
        setSize(400, 400);
        setVisible(true);
        JComboBox JComboBoxObj = new JComboBox();

        JComboBoxObj.addItem("Solapur");
        JComboBoxObj.addItem("Pune");
        JComboBoxObj.addItem("Bangalore");
    }
}
```

```
JComboBoxObj.addItem("Mumbai");
JComboBoxObj.addItemListener(this);

JLabelObj = new JLabel();

add(JComboBoxObj);
add(JLabelObj);
}

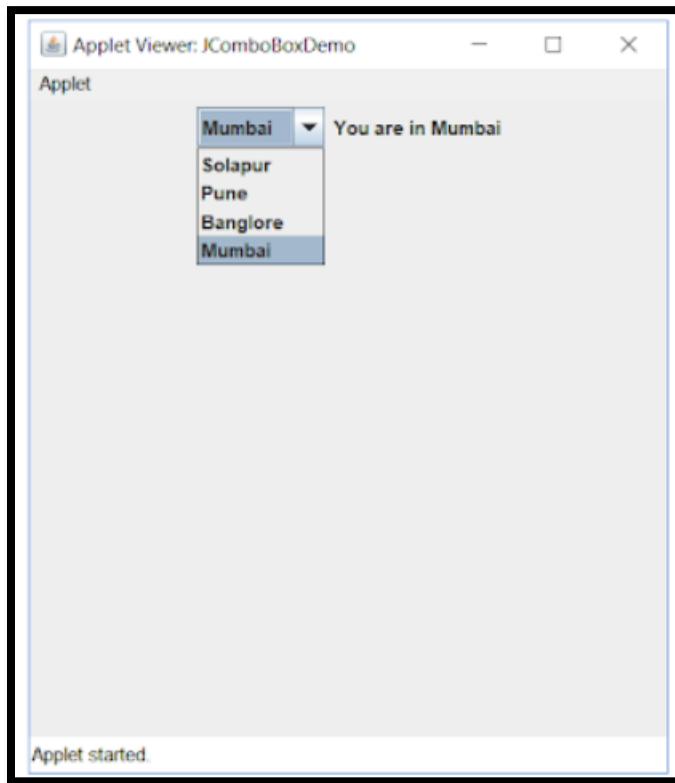
public void itemStateChanged(ItemEvent ie)
{
    String stateName = (String) ie.getItem();

    JLabelObj.setText("You are in "+stateName);
}

}

/* <applet code="JComboBoxDemo" height="400" width="400"> </applet> */
```

Output:



Practical No. 7

Aim :-

Write a Program to create a JTree.

Theory :-

JTree:

JTree is a **Swing** component with which we can display hierarchical data. **JTree** is quite a complex component. A **JTree** has a 'root node' which is the top-most parent for all nodes in the tree. A node is an item in a tree. A node can have many children nodes. These children nodes themselves can have further children nodes. If a node doesn't have any children node, it is called a leaf node.

The leaf node is displayed with a different visual indicator. The nodes with children are displayed with a different visual indicator along with a visual 'handle' which can be used to expand or collapse that node. Expanding a node displays the children and collapsing hides them.

Code:-

```
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;

public class Pract_7 extends JFrame {
    JTree tree;
    Pract_7(){

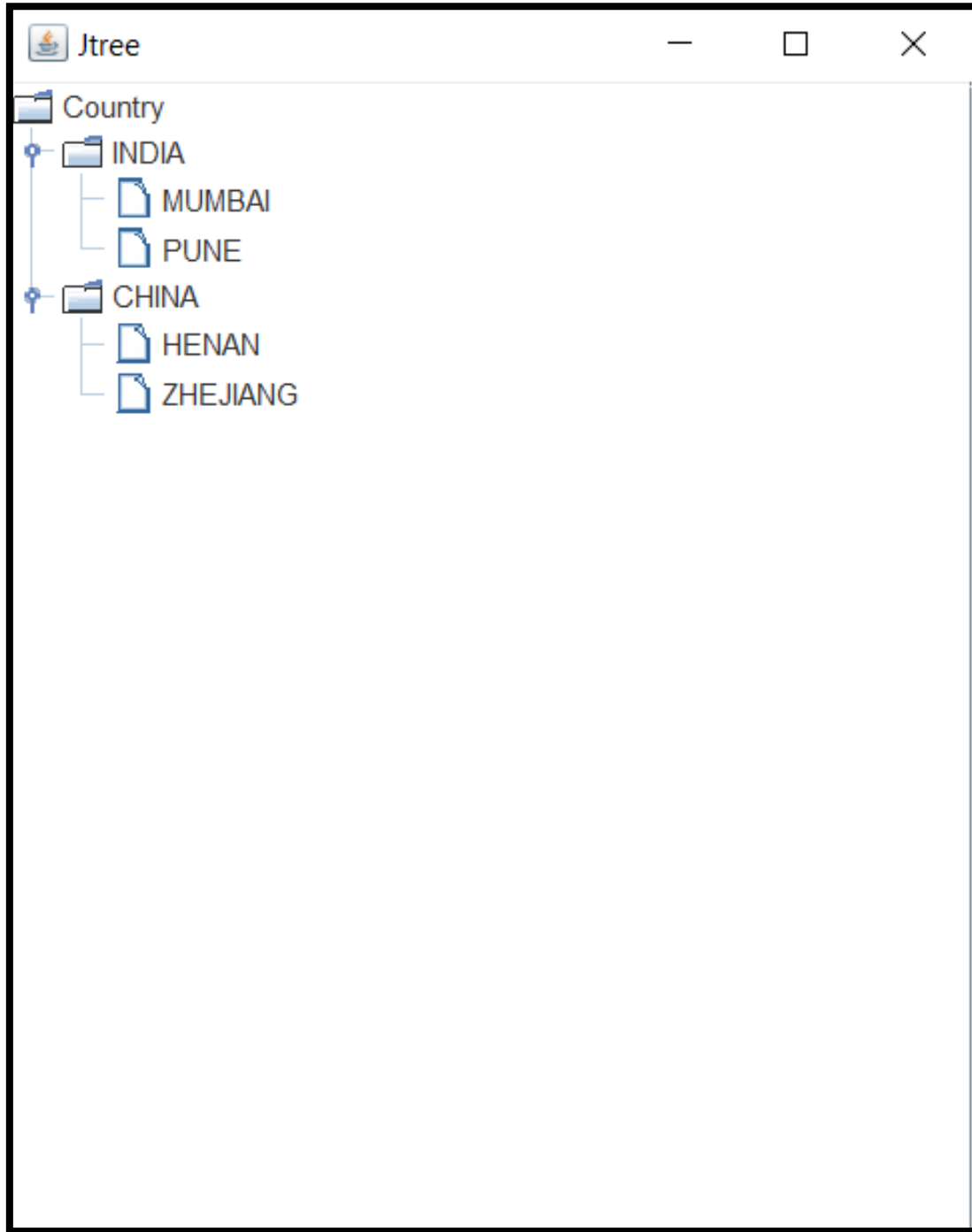
        setLayout(new BorderLayout());
        DefaultMutableTreeNode top = new DefaultMutableTreeNode("Country");
        DefaultMutableTreeNode a = new DefaultMutableTreeNode("INDIA");
        top.add(a);
```

```
DefaultMutableTreeNode a1 = new DefaultMutableTreeNode("MUMBAI");
a.add(a1);
DefaultMutableTreeNode a2 = new DefaultMutableTreeNode("PUNE");
a.add(a2);
DefaultMutableTreeNode b = new DefaultMutableTreeNode("CHINA");
top.add(b);
DefaultMutableTreeNode b1 = new DefaultMutableTreeNode("HENAN");
b.add(b1);
DefaultMutableTreeNode b2 = new DefaultMutableTreeNode("ZHEJIANG");
b.add(b2);

tree = new JTree(top);
JScrollPane sp = new JScrollPane(tree);
add(sp, BorderLayout.CENTER);
setTitle("Jtree");
setSize(400,500);
setVisible(true);

}
public static void main(String args[]) {
    new Pract_7();
}
}
```

Output:



Practical No. 10

Aim :-

Write a program to demonstrate Status of key on applet window such as KeyPressed, KeyReleased, Keyup, KeyDown.

Theory:-

KeyPressed():

This method has been used in the program which receives the generated event when you press any key to the object. Above method also sets the text of the source of the event to the label.

The KeyEvent Class:

A **KeyEvent** is generated when keyboard input occurs. There are three types of key events, which are identified by these integer constants: **KEY_PRESSED**, **KEY_RELEASED**, and **KEY_TYPED**. The first two events are generated when any key is pressed or released. The last event occurs only when a character is generated. Remember, not all keypresses result in characters. For example, pressing shift does not generate a character.

There are many other integer constants that are defined by **KeyEvent**. For example, **VK_0** through **VK_9** and **VK_A** through **VK_Z** define the ASCII equivalents of the numbers and letters. Here are some others:

VK_ALT

VK_CANCEL

VK_CONTROL

VK_DOWN

VK_ENTER

VK_ESCAPE

VK_LEFT

VK_PAGE_DOWN

VK_PAGE_UP

VK_RIGHT

VK_SHIFT

VK_UP

The **VK** constants specify *virtual key codes* and are independent of any modifiers, such as control, shift, or alt.

KeyEvent is a subclass of **InputEvent**. Here is one of its constructors: `KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)`

Here, *src* is a reference to the component that generated the event. The type of the event is specified by *type*. The system time at which the key was pressed is passed in *when*. The *modifiers* argument indicates which modifiers were pressed when this key event occurred. The virtual key code, such as **VK_UP**, **VK_A**, and so forth, is passed in *code*. The character equivalent (if one exists) is passed in *ch*. If no valid character exists, then *ch* contains

CHAR_UNDEFINED.

For **KEY_TYPED** events, *code* will contain **VK_UNDEFINED**. The **KeyEvent** class defines several methods, but probably the most commonly used

ones are **getKeyChar()**, which returns the character that was entered, and **getKeyCode()**, which returns the key code. Their general forms are shown here:

```
char getKeyChar() int getKeyCode()
```

If no valid character is available, then **getKeyChar()** returns **CHAR_UNDEFINED**. When a

KEY_TYPED event occurs, **getKeyCode()** returns **VK_UNDEFINED**.

Code:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class KeyEventDemo extends Applet implements KeyListener
{
    String msg = "";

    public void init()
    {
        addKeyListener(this);
    }

    public void keyReleased(KeyEvent k)
    {
        showStatus("Key Released");
        repaint();
    }

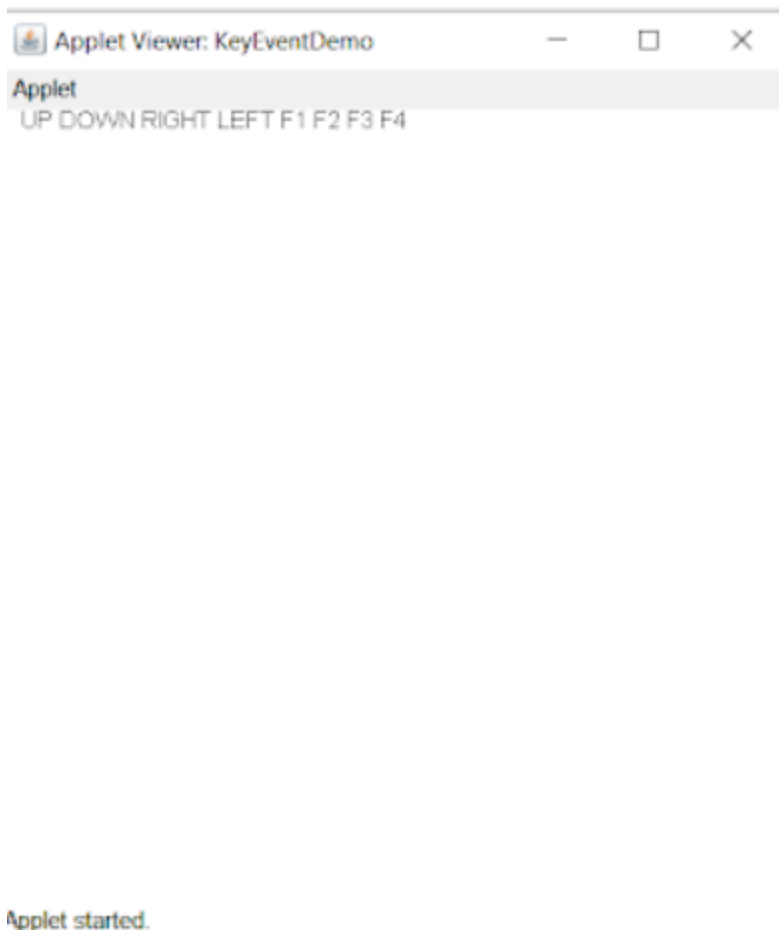
    public void keyTyped(KeyEvent k)
    {
        showStatus("Key Typed");
        repaint();
    }

    public void keyPressed(KeyEvent k)
    {
        showStatus("Key Pressed");
        repaint();
    }

    public void paint(Graphics g)
    {
        g.drawString(msg, 10, 10);
    }
}
```

```
    }  
}  
?  
/*  
    <applet code="KeyEventDemo" height="400" width="400">  
    </applet>  
*/
```

Output:



Practical No. 11

Aim :-

Write a program to demonstrate various Mouse events using `MotionListener` and `MouseMotionListener` interface.

Theory:-

In Java, `MouseListener` is a class that gets notified when there is a change in the mouse state. Changes of the mouse can be pressing, clicking, and releasing it. It can also be entering or exiting the window area. `Mouselistener` is working with the help of keyword `implements` and this listener interface can be gained from the `java.awt.event` package.

Declaration: Java `MouseListener` interface can be declared using the following syntax.

```
Public interface MouseListener extends EventListener
```

Methods of Java `MouseListener`

There are five methods. They are :

- 1. `mouseClicked(MouseEventv)`:** This method will get invoked when a Mouse button is clicked on a component.
- 2. `mouseEntered(MouseEventv)`:** This method will get invoked when a Mouse is entering a component.
- 3. `mouseExited(MouseEventv)`:** This method will get invoked when a Mouse is exiting a component.
- 4. `mousePressed(MouseEventv)`:** This method will get invoked when a Mouse button is pressed on a component.
- 5. `mouseReleased(MouseEventv)`:** This method will get invoked when a Mouse button is released on a component

Code:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class MouseColor extends Applet implements MouseMotionListener
{
    public void init()
    {
        addMouseMotionListener(this);
    }

    public void mouseDragged(MouseEvent me)
    {
        setBackground(Color.red);
        repaint();
    }

    public void mouseMoved(MouseEvent me)
    {
        setBackground(Color.green);
        repaint();
    }
}
/*
<applet code="MouseColor" width=300 height=300>
</applet>
*/
```

Output:



Practical No. 12

Aim :-

Write a program to demonstrate the use of JTextField and JPasswordField using listener interface.

Theory:-

JTextField is a part of javax.swing package. The class JTextField is a component that allows editing of a single line of text. JTextField inherits the JTextComponent class and uses the interface SwingConstants. The constructor of the class are :

1. **JTextField()** : constructor that creates a new TextField
2. **JTextField(int columns)** : constructor that creates a new empty TextField with specified number of columns.
3. **JTextField(String text)** : constructor that creates a new empty text field initialized with the given string.
4. **JTextField(String text, int columns)** : constructor that creates a new empty textField with the given string and a specified number of columns.
5. **JTextField(Document doc, String text, int columns)** : constructor that creates a textField that uses the given text storage model and the given number of columns.

Methods of the JTextField are:

1. **setColumns(int n)** :set the number of columns of the text field.
2. **setFont(Font f)** : set the font of text displayed in text field.
3. **addActionListener(ActionListener l)** : set an ActionListener to the text field.
4. **int getColumns()** :get the number of columns in the textfield.

PasswordField is a part of javax.swing package . The class JPasswordField is a component that allows editing of a single line of text where the view indicates that something was typed by does not show the actual characters. JPasswordField inherits the JTextField class in javax.swing package.

JPasswordField

Constructors of the class are :

1. **JPasswordField()**: constructor that creates a new PasswordField
2. **JPasswordField(int columns)** : constructor that creates a new empty PasswordField with specified number of columns.
3. **JPasswordField(String Password)** : constructor that creates a new empty Password field initialized with the given string.
4. **JPasswordField(String Password, int columns)** : constructor that creates a new empty PasswordField with the given string and a specified number of columns .
5. **JPasswordField(Document doc, String Password, int columns)** : constructor that creates a Passwordfield that uses the given text storage model and the given number of columns.

Code:-

```
import javax.swing.*;
import java.awt.*;

public class JPasswordFieldChange
{
    public static void main(String[] args) {
        JFrame f = new JFrame();

        f.setVisible(true);
        f.setSize(400,400);
        f.setLayout(new FlowLayout());

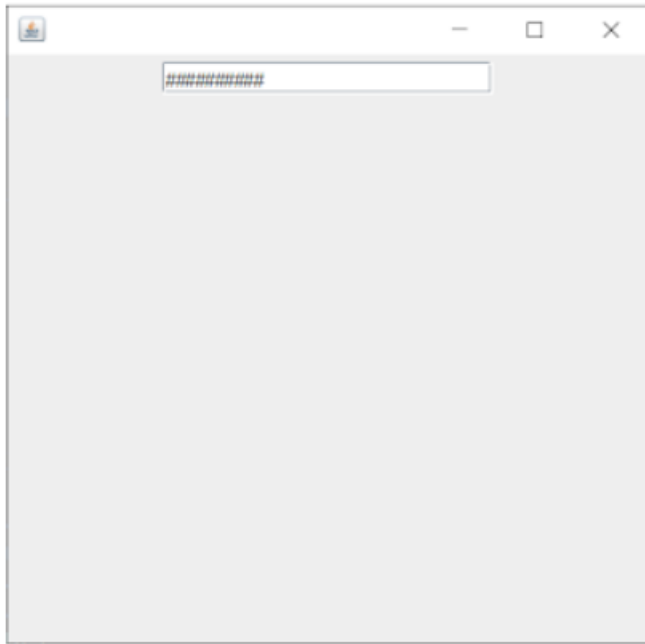
        JPasswordField pf = new JPasswordField(20);

        pf.setEchoChar('#');

        f.add(pf);
    }
}
```

```
}
```

Output:-



JTextField

Code:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class JTextAdd implements ActionListener
{
    JTextField tf , tf1 ;
    JLabel res;

    JTextAdd()
    {
        JFrame f = new JFrame();
```



```
f.setVisible(true);
f.setSize(400,400);
f.setLayout(new FlowLayout());

JLabel jl = new JLabel("Enter 1st Number:");
tf = new JTextField(5);
JLabel jl1 = new JLabel("Enter 2nd Number:");
tf1 = new JTextField(5);
res = new JLabel("Addition");

tf1.addActionListener(this);

f.add(jl);
f.add(tf);
f.add(jl1);
f.add(tf1);
f.add(res);

}

public static void main(String[] args) {
    JTextAdd jt = new JTextAdd();
}

public void actionPerformed(ActionEvent ae)
{
    String str1 = tf.getText();
    double fn = Double.parseDouble(str1);
    double sn = Double.parseDouble(tf1.getText());

    res.setText("Sum is " + (fn+sn));
}
}
```

Output:

