10. USING AWT IN APPLICATION

❖ The AWT classes

- The Abstract Window Toolkit(AWT) contains numerous classes and methods that allow you to create and manage windows.
- Although the main purpose of the AWT is to support applet windows, it can also be used to create stand-alone windows that run in GUI environment, such as windows.
- The AWT classes are contained in the 'java.awt' package.
- Fortunately, because it is logically organized in a top-down, hierarchical fashion, it is easier to understand and use.
- **Following are the list of some AWT classes.**

Class	Description
AWTEvent	Encapsulates AWT events.
BorderLayout	The border layout manager. Border layouts use five components: North, South, East West, and Center.
Button	Creates a push button control.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index cards. Only the one on top is showing.
Checkbox	Creates a checkbox control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list/Dropdown list(Combo box).
Color	Manages colors in a portable, platform-independent fashion.
Container	A subclass of Component that can hold other components.
Dialog	Creates a dialog window.
Event	Encapsulates events.
FileDialog	Creates a window from which a file can be selected.
FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Font	Encapsulates a type font.
Frame	Creates a standard window that has t title bar, resize corners, and a menu bar.
Graphics	Encapsulates the graphics context. This context is used by the various output methods to display output in a window.
GridLayout	The grid layout manager. Grid layout displays components in a two-dimensional grid.
Image	Encapsulates graphical images.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose. Similar to the standard Windows list box.
Menu	Creates a pull-down menu.
MenuBar	Creates a menu bar.
MenuComponent	An abstract class implemented by various menu classes.
MenuItem	Creates a menu item.
MenuShortcut	Encapsulates a keyboard shortcut for a menu item.
Panel	The simplest concrete subclass of Container.

Point	Encapsulates a Cartesian co-ordinate pair, stored in x and y.
Polygon	Encapsulates a polygon.
PopupMenu	Encapsulates a pop-up menu.
PrintJob	An abstract class that represents a print job.
Rectangle	Encapsulates a rectangle.
Scrollbar	Creates a scroll bar control.
ScrollPane	A container that provides horizontal and/or vertical scroll bars for another component.
SystemColor	Contains the colors of GUI widgets such as windows, scroll bars, text, and others.
TextArea	Creates a multiline edit control.
TextComponent	A superclass for TextArea and TextField.
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar, and no title.

❖ Layout Manager:

- A layout manager is an object that is used to organize components in a container.
- Layout managers are objects that determine how the components are organized in a container.
- Every container has a default layout manager.
- **There are types of layout manager:**
  - **Flow layout**
  - **Grid layout**
  - **Border layout**
  - **Card layout**
- A class of the same name represents each layout manager.
- To use a layout manager in a Java program, you need to use an object of the corresponding layout manager class.
- To use a layout manager for a container, create an object of the appropriate layout manager class, and use the setLayout ( ) method to specify the type of the layout to be used.

➢ Flow Layout

- The flow layout manager is the default layout manager for applets and panels.
- It places controls linearly from left to right and from top to bottom in horizontal and vertical rows.
- For example, when the layout manager reaches the right border of the container, it positions the components in the next row.
- The flow layout manager places the controls centrally aligned in each row.
- **Example:-**

```
import java.applet.*;
import java.awt.*;
public class Aptflow1 extends Applet
{
        public void init()
        {
                /*
```

```
                    <Applet CODE="Aptflow1.class" width="100" height="100">
                    </Applet>
                    */
                    Button bt1=new Button("ok");
                    Button bt2=new Button("save");
                    FlowLayout bl=new FlowLayout(FlowLayout.LEFT);
                    setLayout(bl);
                    add(bt1);
                    add(bt2);
            }
    }
```

- **Grid Layout**
  - The grid layout manager is used to position components in a grid.
  - The grid layout manager organizes the components by placing them in a rectangular grid consisting of rows and columns.
  - The number of components and the size of the container determine the size of a cell.
  - The manager divides the area of the container by the number of components to obtain the size of a cell.
  - **Example:-**

```
import java.applet.*;
import java.awt.*;
public class Aptgrid extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="Aptgrid.class" width="100" height="100">
                </Applet>
                */
                Button bt1=new Button("ok");
                Button bt2=new Button("save");
                Button bt3=new Button("ok1");
                Button bt4=new Button("save1");
                GridLayout gl=new GridLayout(2,2);
                setLayout(gl);
                add(bt1);
                add(bt2);
                add(bt3);
                add(bt4);
        }
}
```

- **Border Layout**
  - The border layout is the default layout for windows and dialog boxes.
  - The border layout manager is used to organize components along the four borders of the container with one component in the center.
  - In addition, the border layout manager is used to create interfaces that focus on region- based positioning(directions) of components instead of coordinates based position as in flow and layouts, and to create dialog boxes that display captions along with the 'OK' and 'CANCEL' buttons.
  - **Example:-**

```
import java.applet.*;
```

```
import java.awt.*;
public class AptBorder extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="AptBorder.class" width="100" height="100">
                </Applet>
                */
                Button bt1=new Button("ok");
                Button bt2=new Button("cancel");
                BorderLayout bl=new BorderLayout(2,2);
                setLayout(bl);
                add(bt1,BorderLayout.NORTH);
                add(bt2,BorderLayout.SOUTH);
        }
}
```

- **Card Layout**
  - The CardLayout layout manager is significantly different from the other layouts.
  - The CardLayout is unusual.
  - It breaks the applet into a deck of cards, each of which will normally have a panel with its own LayoutManager.
  - Only one card appears on the screen at a time.
  - You then flip between cards, each of which shows a different set of components.
  - The common analogy is with an installer wizard.
  - This might be used for a series of data input screens, where more input is needed than can comfortably be fit on one screen.
  - Conversely, you can use a CardLayout for a slide show, where there's more data to be presented than will fit on one screen.
  - **Example:**

```
import java.awt.*;
import java.applet.*;
public class CardExample extends Applet
{
        /*
        <Applet CODE="CardExample.class" width="100" height="100">
        </Applet>
        */
        CardLayout cl=new CardLayout();
        public void init()
        {
                setLayout(cl);
                Panel pA = new Panel();
                Panel pC = new Panel();
                setLayout(new GridLayout(3,1));
                List l =new List(3,false);
                Choice c=new Choice();
                l.add("A");
                l.add("B");
                l.add("C");
```

```
                    c.add("Male");
                    c.add("Female");

                    pA.add(l);
                    pC.add(c);
                    add("One",pA);
                    add("Two",new Button("Click Here"));
                    add("Three",pC);
                }
        }
```

- **Panel:**
  - To make Applet or Frame layout easier, you break a frame up into regions and compose each of them separately. Each region is called a Panel
  - Each can have its own different LayoutManager. Panels don't have any visible bounding lines.
  - You can delimit them with differing background colours.
  - **Example:**

```
import java.awt.*;
import java.applet.*;
public class PanelExample extends Applet
{
        /*
        <Applet CODE="PanelExample.class" width="100" height="100">
        </Applet>
        */
        public void init()
        {
                Button bt1=new Button("ok");
                Panel p1=new Panel();
                p1.add(bt1);
                add(p1);
        }
}
```

- **Canvas**
  - The java.awt.Canvas class is a rectangular area on which you can draw using the methods of java.awt.Graphics.
  - The Canvas class has only three methods:
    - ***public Canvas()***
    - ***public void addNotify()***
    - ***public void paint(Graphics g)***

Example:-

```
import java.awt.*;
import java.applet.*;
 public class MyCanvas extends Applet
 {
  /*<applet code=MyCanvas.class  height=300 width=500>
  </applet>*/
  public MyCanvas()
  {
    setSize(200, 400);
  }
  public void paint(Graphics g)
  {
    g.drawRect(0, 0, 90, 50);
    g.drawString("A Canvas", 15,15);
  } }
```

- You generally won't instantiate a canvas directly. Instead you'll subclass it and override the paint() method in your subclass to draw the picture you want.

❖ **Frame**

- **Usage:** A Frame is a top-level window with a title and a border. It is normally used as the main window of a standalone application.
- Package: java.awt
- Class: java.awt.Frame
- **Parent Classes**
  - java.awt.Frame implements MenuContainer
  - java.awt.Window implements Accessible
  - java.awt.Container
  - java.awt.Component implements ImageObserver, MenuContainer, Serializable
  - java.lang.Object
- **Common Public Methods**
  - **addWindowListener(WindowListener Handler):** Configures a window event handler for the frame.
  - **setBackground (Color BackgroundColor):** Sets the background color of the frame.
  - **setFont (Font TextFont):** Sets the font for this component.
  - **setForeground (Color TextColor):** Sets the color of the text for the frame.
  - **setSize (int Width, int Height):** Resizes this window so that it has the specified Width and Height.
  - **setTitle (String Text):** Sets the text for the title bar.
  - **show ():** Makes the window visible.
- **Arguments**
  - BackgroundColor**:** The color to be used for the background of the button.
  - Handler**:** The object which handles window events from this frame.
  - Text**:** The text to appear in the title bar.
  - TextColor**:** The color to be used for the text of the button.
  - TextFont**:** The font to be used for the text of the button.

- **Example:**

```
import java.awt.*;
import java.awt.event.*;
public class Frame2 extends Frame
{
   public static void main(String[] args)
   {
         Frame f=new Frame("Button Frame");
        Button button = new Button("Submit");
        f.add(button);
        f.setLayout(new FlowLayout());
        f.setSize(200,100);
        f.setVisible(true);

        f.addWindowListener(new WindowAdapter()
        {
              public void windowClosing(WindowEvent e)
              {
                    System.exit(0);
               }
        }
        );
 } }
```

❖ Dialog box

- **Dialog boxes** are **pop-up** windows on the screen that appear for a small time to take either input or display output while a main application is running
- The windows that appear for a brief time are known as **transient windows**.
- Dialog boxes are generally used to draw special attention of the user like displaying warnings.
- Dialog box is a **top-level** window that comes with a border including a title bar.
- The dialog box can be made non-resizable and the default layout manager is **BorderLayout** which can be overridden.
- A dialog box works within a main program.
- It cannot be created as a standalone application.
- It is a **child window** and must be connected to a main program or to a **parent window**.
- Frame is a parent window as it can work independently.
- For example, the **Find and Replace Dialog box** cannot be obtained without opening MS-Word document.
- Likewise, **File Deletion Confirmation box** cannot appear without deleting a file.
- Two commonly used constructors are shown here:
  - Dialog(Frame *parentWindow*, boolean *mode*)
  - Dialog(Frame *parentWindow*, String *title*, boolean *mode*)
- **Example:**

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

public class test extends Applet implements ActionListener
{
        /*<applet code="test.class" width="320" height="230" ></applet>*/

        public void init()
        {
                Button bt1 = new Button("Push Button");
                bt1.addActionListener(this);
                add(bt1);
        }
        public void actionPerformed(ActionEvent e)
        {
                Frame f = new Frame();
                Dialog d = new Dialog(f, "hello", true);
                d.add(new Label("hello"));
                d.show();
        }
}
```

❖ Controls

❖ Textbox

- Usage: The **TextField** class can be used to add single line text input to a container.
- Package: Java.awt
- **Parent Classes:**
  - java.awt.TextComponent implements Accessible
  - java.awt.Component implements ImageObserver, MenuContainer, Serializable
  - java.lang.Object

- **Common Public Methods**
  - **void addActionListener (ActionListener Handler):**Configures an event handler for the TextField.
  - **String getText():-** Returns the text in the field.
  - **void setBackground(Color.BackgroundColor):-** Sets the background color of the TextField.
  - **void setEditable (boolean Editable):-** Sets the field as being editable or fixed.
  - **void setFont (Font TextFont):-** Sets the font for this component.
  - **void setText (String Text):-** Sets the text for the field.
- **Arguments**
  - BackgroundColor**:-** The color to be used for the background of the field.
  - Editable**:-** The state of the field as editable or fixed.
  - Handler**:-** The object which handles action events from this field.
  - Text**:-** The text to appear in the field.
  - TextFont**:-** The font to be used for the text of the button.

- **Example:**

```
import java.applet.*;
import java.awt.*;
public class AptTextbox extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="AptTextbox.class" width="100" height="100">
                </Applet>
                */
                TextField t=new TextField("Your Name");
                add(t);
        }
}
```

❖ TextArea

- **Usage:** The **TextArea** class is a GUI component which can be used to provide multiple line text input or output.
- Package: Java.awt
- **Parent Classes:**
  - java.awt.TextComponent implements Accessible
  - java.awt.Component implements ImageObserver, MenuContainer, Serializable
  - java.lang.Object
- **Common Public Methods**
  - **void addActionListener (ActionListener Handler):**Configures an event handler for the TextField.
  - **String getText ():-** Returns the text in the field.
  - **void setBackground (Color BackgroundColor):-** Sets the background color of the TextField.
  - **void setEditable (boolean Editable):-** Sets the field as being editable or fixed.
  - **void setFont (Font TextFont):-** Sets the font for this component.
  - **void setText (String Text):-** Sets the text for the field.
- **Arguments**
  - BackgroundColor**:-** The color to be used for the background of the field.
  - Editable**:-** The state of the field as editable or fixed.
  - Handler**:-** The object which handles action events from this field.

- Text**:-** The text to appear in the field.
- Rows**:-** The height of the text area.
- TextFont**:-** The font to be used for the text of the button.

➢ **Example:-**

```
import java.applet.*;
import java.awt.*;
public class AptTextArea extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="AptTextArea.class" width="100" height="100">
                </Applet>
                */
                TextArea t=new TextArea("Your Address");
                add(t);
        }
}
```

❖ **Button**

➢ Usage:- The Button class can be used to add interactive buttons to a container.

➢ Package:- Java.awt

➢ **Parent Classes**

- java.awt.Component implements ImageObserver, MenuContainer, Serializable
- java.lang.Object

➢ **Common Public Methods**

- **void addActionListener (ActionListener Handler)**
- **void setBackground (Color BackgroundColor):-** Sets the background color of the button.
- **void setEnabled (boolean State):-** Enables or disables the button.
- **void setFont (Font TextFont):-** Sets the font for this component.
- **void setForeground (Color TextColor):-** Sets the color of the text for the button.
- **void setLabel (String Text):-** Sets the text for the button.

➢ **Arguments**

- ActionText**:-** The ActionCommand text for the event associated with the button.
- BackgroundColor**:-** The color to be used for the background of the button.
- Handler**:-** The object which handles action events from this combo box.
- State**:-** The enable state of the button (true or false.)
- Text**:-** The text to appear next to the button.
- TextColor**:-** The color to be used for the text of the button.
- TextFont**:-** The font to be used for the text of the button.

➢ **Example:**

```
import java.applet.*;
import java.awt.*;
public class AptButton extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="AptButton.class" width="100" height="100">
                </Applet>
```

```
                    */
                    Button b1=new Button("Submit");
                    add(b1);
                }
        }
```

- **Label**
  - Usage: The Label class can be used to add text labels to a container.
  - Package: **Java.awt**
  - Subclass of: Java.awt.Component
  - **Common Methods:**
    - **void setBackground (Color BgdColor):**Sets the color of the background for the label.
    - **void setAlignment (int Alignment):**Sets the alignment of the label's contents along the X axis.
    - **void setFont (Font TextFont):**Sets the font for this component.
    - **void setForeground (Color TextColor):**Sets the color of the text for the label.
    - **void setText (String Text):**Sets the text for the label.
  - **Arguments:**
    - **Alignment:** The horiziontal position of the text within the label. One of:
      - Label.CENTER: The central position in an area.
      - Label.LEFT: Box-orientation constant used to specify the left side of a box.
      - Label.RIGHT: Box-orientation constant used to specify the right side of a box.
    - **BgdColor:**The color to be used for the background of the label.
    - **Text:**The text to appear in the label.
    - **TextColor:**The color to be used for the text of the label.
    - **TextFont:**The font to be used for the text of the label.
  - **Example:**

```
import java.applet.*;
import java.awt.*;
public class Aptlabel extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="Aptlabel.class" width="100" height="100">
                </Applet>
                */
                Label l1=new Label("User Name");
                add(l1);
        }
}
```

- **CheckBox**
  - **Usage:The CheckBox class can be used to add interactive grouped radio checkboxs to a container.**
  - Package: **Java.awt**
  - Subclass of: Java.awt.Component
  - **Common Methods:**
    - **void addItemListener (ItemListener Handler):-** Configures an event handler for the checkbox.
    - **boolean isSelected ():-** Returns the state (checked or not checked) of the checkbox.

- **void setBackground (Color BackgroundColor):-** Sets the background color of the checkbox.
- **void setSelected (boolean State):-** Sets the state (checked or not checked) of the checkbox.
- **void setFont (Font TextFont):-** Sets the font for this component.
- **void setForeground (Color TextColor):-** Sets the color of the text for the checkbox.
- **void setLabel (String Text):-** Sets the text for the checkbox.

➢ **Arguments:**

- **BackgroundColor:-** The color to be used for the background of the checkbox.
- **Group:-** A CheckboxGroup for interactive boxes to act as radiobuttons.
- **Handler:-** The object which handles action events from this combo box.
- **State:-** The checked state of the checkbox (true or false.)
- **Text:-** The text to appear in the checkbox.
- **TextColor:-** The color to be used for the text of the checkbox.
- **TextFont:-** The font to be used for the text of the checkbox.

➢ **Example: 1(Simple CheckBox)**

```
import java.applet.*;
import java.awt.*;
public class AptCheckbox extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="AptCheckbox.class" width="100" height="100">
                </Applet>
                */
                Checkbox ch=new Checkbox("FootBall");
                add(ch);
                Checkbox ch1=new Checkbox("Cricket");
                add(ch1);
        }
}
```

➢ ***Example:2(RadioButton)***

```
import java.applet.*;
import java.awt.*;
public class AptRadio extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="AptRadio.class" width="100" height="100">
                </Applet>
                */
                CheckboxGroup cg =new CheckboxGroup();
                Checkbox cmale=new Checkbox("Male",cg,false);
                add(cmale);
                Checkbox cfemale=new Checkbox("Female",cg,false);
                add(cfemale);
        }
}
```

❖ Choice

- Usage: The Choice(Combobox) class can be used to add a drop-down list of items to a container
- Package: **Java.awt**
- Subclass of: Java.awt.Component
- **Common Methods:**
  - **void addItemListener (ItemListener Handler):-** Configures an event handler for the choice box.
  - **void insert (String Item, int Index):** Inserts the item into this choice at the specified position.
  - **int getSelectedIndex ():** Returns the index of the selected item.
  - **void setBackground (Color BackgroundColor):** Sets the background color of the choice box.
  - **void setFont (Font TextFont):** Sets the font for this component.
  - **void setForeground (Color TextColor):** Sets the color of the text for the choice box.
  - **void Select (int Selection):** Sets the index of the selected item.
- **Arguments:**
  - **BackgroundColor:** The color to be used for the background of the choice box.
  - **Index:** Position where the item will be inserted.
  - **Item:** A String to be listed in the choice box.
  - **Handler:** The object which handles action events from this choice box.
  - **Selection:** The index of the selected item.
  - **TextColor:** The color to be used for the text of the choice box.
  - **TextFont:** The font to be used for the text of the choice box.
- **Example:-**

```
import java.applet.*;
import java.awt.*;
public class AptChoice extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="AptChoice.class" width="100" height="100">
                </Applet>
                */
                Choice ch=new Choice();
                ch.add("India");
                ch.add("USA");
                add(ch);
        }
}
```

❖ List

- Usage: The List class can be used to add a list of items with several visible rows to a container.
- Package: **java.awt**
- Class**: java.awt.List**
- Parent Classes
  - java.awt.List
  - java.awt.Component
  - java.lang.Object
- **Common Public Methods**
  - **void addItemListener (ItemListener Handler) :** Specifies an event handler for the list.
  - **void add (String Item) :** Inserts the item into this choice at the next position.
  - **void add (String Item, int Index) :** Inserts the item into this choice at the specified position.

- **int getSelectedIndex ():** Returns the index of the selected item.
- **void setBackground (Color BackgroundColor):** Sets the background color of the list.
- **void setFont (Font TextFont) :** Sets the font for this component.
- **void setForeground (Color TextColor) :** Sets the color of the text for the list.
- **void Select (int Selection):** Sets the index of the selected item.

➢ **Arguments**
- **BackgroundColor:** The color to be used for the background of the list.
- **Index:** Position where the item will be inserted.
- **Item:** A String to be listed in the list.
- **Handler:** The object which handles action events from this list.
- **Multiselect:** If true, multiple items can be selected.
- **Selection:** The index of the selected item.
- **TextColor:** The color to be used for the text of the list.
- **TextFont:** The font to be used for the text of the list.

➢ **Example:-**

```
import java.applet.*;
import java.awt.*;
public class AptList extends Applet
{
        public void init()
        {
                /*
                <Applet CODE="AptList.class" width="100" height="100">
                </Applet>
                */
                List  lt=new List();
                lt.add("A");
                lt.add("B");
                lt.add("C");
                add(lt);
        }
}
```

❖ **Scrollbar**

➢ Usage: The Scrollbar lets the user graphically select a value by sliding a knob within a bounded interval.
➢ Package: **Java.awt**
➢ Subclass of: Java.awt.Component
➢ **Common Methods:**
- **addAdjustmentListener (AdjustmentListener Handler):** Configures an event handler for the scrollbar.
- **getValue ():** Returns the value of the scrollbar setting.
- **setBackground (Color BackgroundColor):** Sets the background color of the scrollbar.
- **setMaximum (int Max):** Sets the maximum value of the scrollbar.
- **setMinimum (int Min):** Sets the minimum value of the scrollbar.
- **setValue (int Value):** Sets the current value of the scrollbar.

➢ **Arguments:**
- **BackgroundColor:** The color to be used for the background of the field.
- **Handler:** The object which handles action events from this field.
- **Max:** The maximum value of the scrollbar.
- **Min:** The minimum value of the scrollbar.

- **Orientation:** The orientation of the scrollbar.
  - Scrollbar.HORIZONTAL**:** Orients the scrollbar horizontally.
  - Scrollbar.VERTICAL**:** Orients the scrollbar vertically.
- **Value:** The current value of the scrollbar.

➢ **Example:**

```
import java.applet.*;
import java.awt.*;
public class  AptScroll extends Applet
{
        /*
        <Applet CODE="AptScroll.class" width="100" height="100">
        </Applet>
        */
        Scrollbar vs,hs;
        public void init()
        {
                vs = new Scrollbar(Scrollbar.VERTICAL);
                hs = new Scrollbar(Scrollbar.HORIZONTAL);
                add(vs);
                add(hs);
        }
}
```

❖ Difference between Applet and Stand-alone Application

- An applet is a program written in the Java programming language that can be included in an HTML page, much in the same way an image is included. An application is a standalone Java program that runs as a true application, outside of a browser. Both require a JVM (Java Virtual Machine).
- An applet runs under the control of a browser, whereas an application runs stand-alone, with the support of a virtual machine. As such, an applet is subjected to more stringent security restrictions in terms of file and network access, whereas an application can have free reign over these resources.
- Applets are great for creating dynamic and interactive web applications, but the true power of Java lies in writing full blown applications. With the limitation of disk and network access, it would be difficult to write commercial applications (though through the user of server based file systems, not impossible). However, a Java application has full network and local file system access, and its potential is limited only by the creativity of its developers

❖ Extra Example:

Example 1: Write a program to Scroll the scrollbar and value display in textbox.

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

public class AptScroll extends Applet implements AdjustmentListener
{
        /*
        <Applet CODE="AptScroll.class" width="100" height="100">
        </Applet>
        */
        Scrollbar vs,hs;
        TextField t1;
```

```
        TextField t2;
        public void init()
        {
                vs = new Scrollbar(Scrollbar.VERTICAL);
                t1 = new TextField(2);
                hs = new Scrollbar(Scrollbar.HORIZONTAL);
                t2 = new TextField(2);
                add(vs);
                add(t1);
                add(hs);
                add(t2);
                vs.addAdjustmentListener(this);
                hs.addAdjustmentListener(this);
        }

        public void adjustmentValueChanged(AdjustmentEvent ae)
        {
                int val = hs.getValue();
                t1.setText(String.valueOf(val));

               int val1 = vs.getValue();
              t2.setText(String.valueOf(val1));
        }

}
```

❖ **Example 2: Write a program using ActionListener Event of a button.**

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class Actionevent extends Applet implements ActionListener
{
        public void init()
        {
                /*
                <Applet CODE="Actionevent.class" width="100" height="100">
                </Applet>
                */
                Button bt1=new Button("ok");
                add(bt1);
                bt1.addActionListener(this);
        }
        public void actionPerformed(ActionEvent e)
        {
                setBackground(Color.red);
        }
}
```

❖ **Example 3: Write a program to add two numbers using Textbox.**

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class AptSum extends Applet implements ActionListener
{
    public int  c;
    TextField t1,t2,t3;
    public void init()
   {
            // <Applet CODE="AptSum.class" width="100" height="100"></Applet>
            Button bt=new Button("Sum");
            add(bt);
            t1=new TextField(20);
            add(t1);
            t2=new TextField(20);
            add(t2);
            t3=new TextField(20);
            add(t3);
            bt.addActionListener(this);
   }
    public void actionPerformed(ActionEvent e)
    {
            c=Integer.parseInt(t1.getText())+Integer.parseInt(t2.getText());
            t3.setText(" " +c);
    }
}
```

❖ **Example 4: Write a program using Mouse Event.**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public  class AptMouse extends Applet implements MouseListener,MouseMotionListener
{
        // <Applet CODE="AptMouse.class" width="100" height="100"></Applet>
        public void init()
        {
                addMouseListener(this);
                addMouseMotionListener(this);
        }
        public void mouseClicked(MouseEvent me)
        {
                setBackground(Color.red);
        }
        public void mouseEntered(MouseEvent me)
        {
                setBackground(Color.blue);
        }
        public void mouseExited(MouseEvent me)
        {
```

```
                setBackground(Color.green);
        }
        public void mousePressed(MouseEvent me)
        {
                setBackground(Color.pink);
        }
        public void mouseReleased(MouseEvent me)
        {
                setBackground(Color.black);
        }
        public void mouseDragged(MouseEvent me)
        {
                setBackground(Color.red);
        }
        public void mouseMoved(MouseEvent me)
        {
                setBackground(Color.red);
        }
}
```

❖ **Example 5: Write a program using FocusListener event.**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class AptFocus extends Applet implements FocusListener
{
    TextField t1,t2,t3,t4,t5;
    public void init()
    {
            //  <Applet CODE="AptFocus.class" width="100" height="100"></Applet>
            t1=new TextField(12);
            t2=new TextField(13);
            t3=new TextField(14);
            t4=new TextField(15);
            t5=new TextField(15);
            add(t1);
            add(t2);
            add(t3);
            add(t4);
            add(t5);
            t3.addFocusListener(this);
    }
    public void focusGained(FocusEvent e)
    {
            int a,b,c;
            a=Integer.parseInt(t1.getText());
            b=Integer.parseInt(t2.getText());
            c=Integer.parseInt(t3.getText());
            if(a<b && a<c)
            {
                    t5.setText(" "+a);
```

```
            }
            else if(b<a && b<c)
            {
                  t5.setText(" " +b);
            }
            else
            {
                  t5.setText(" " +c);
            }
      }
      public void focusLost(FocusEvent e)
      {
            int a,b,c;
            a=Integer.parseInt(t1.getText());
            b=Integer.parseInt(t2.getText());
            c=Integer.parseInt(t3.getText());
            if(a>b && a>c)
            {
                  t4.setText(" "+a);

            }
            else if(b>a && b>c)
            {
                  t4.setText(" " +b);
            }
            else
            {
                  t4.setText(" " +c);
            }
      }
   }
```

❖ **Example 6: Write a program using ItemListener event**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class AptItem extends Applet implements ItemListener
{
      //<Applet CODE="AptItem.class" width="100" height="100"></Applet>
      public void init()
      {
            Checkbox ch1=new Checkbox("Red");
            add(ch1);
            ch1.addItemListener(this);
      }
      public void itemStateChanged(ItemEvent e)
      {
            setBackground(Color.red);
      }
}
```

❖ **Example 7: Write a program of Applet life cycle.**

```
import java.applet.*;
import java.awt.*;
public class AptCycle extends Applet
{
        // <Applet CODE="AptCycle.class" width="100" height="100">        </Applet>
        String msg;
        public void init()
        {
                msg="inti";
        }
        public void start()
        {
                msg=msg + "start";
        }
        public void paint(Graphics g)
        {
                g.drawString(msg,10,10);
        }
        public void stop()
        {
                System.out.println("stop");
        }
        public void destroy()
        {
         System.out.println("destroy");
        }
}
```

❖ **Example 8: Write a program of calculator using applet.**

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

public class AptCal extends  Applet implements ActionListener
{
        // <applet code=AptCal.class height=400 width=500></applet>
        TextField t1,t2,t3;
        Button bt1,bt2,bt3,bt4;
        public void init()
        {
          t1=new TextField(15);
          t2=new TextField(15);
          t3=new TextField(15);

          bt1=new Button("Add");
          bt2=new Button("Sub");
          bt3=new Button("Mul");
          bt4=new Button("Div");
```

```
            add(bt1);
            add(bt2);
            add(bt3);
            add(bt4);
            add(t1);
            add(t2);
            add(t3);

            bt1.addActionListener(this);
            bt2.addActionListener(this);
            bt3.addActionListener(this);
            bt4.addActionListener(this);
        }
        public void actionPerformed(ActionEvent e)
        {
                int a,b,c;
                a=Integer.parseInt(t1.getText());
                b=Integer.parseInt(t2.getText());
                if(e.getSource()==bt1)
                {
                        c=a+b;
                        t3.setText(" "+ c);
                }
                else if(e.getSource()==bt2)
                {
                        c=a-b;
                        t3.setText(" "+ c);
                }
                else if(e.getSource()==bt3)
                {
                        c=a*b;
                        t3.setText(" "+ c);
                }
                else
                {
                        c=a/b;
                        t3.setText(" "+ c);
                }
        }
    }
```

❖ **Example 9: Write a program to implement Menu Bar using Frame.**

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

public class MainWindow1 extends Frame
{
    public MainWindow1()
    {
        super("Menu Window");
        setSize(400, 400);
```

```
        FileMenu fileMenu = new FileMenu(this);
        MenuBar mb = new MenuBar();
        mb.add(fileMenu);
        setMenuBar(mb);
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                exit();
            }
        }
        );
    }
     public void exit()
    {
        setVisible(false);
        dispose();
        System.exit(0);
    }
     public static void main(String args[])
    {
        MainWindow1 w = new MainWindow1();
       w.setVisible(true);
    }
   class FileMenu extends Menu implements ActionListener
  {
         MainWindow1 mw;
       public FileMenu(MainWindow1 m)
       {
            super("File");
            mw = m;
            MenuItem mi;
            add(mi = new MenuItem("Open"));
            mi.addActionListener(this);
           add(mi = new MenuItem("Close"));
           mi.addActionListener(this);
            add(mi = new MenuItem("Exit"));
           mi.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e)
    {
        String item = e.getActionCommand();
        if (item.equals("Exit"))
            mw.exit();
        else
            System.out.println("Selected FileMenu " + item);
    }
  }
   }
```

❖ **Example 10: Write a program using KeyListener event**

```
public class keyboard extends Applet implements KeyListener
{
    //<Applet CODE="keyboard.class" width="100" height="100">        </Applet>

   String msg="";
    public void init()
    {
        addKeyListener(this);
    }
    public void keyPressed(KeyEvent ke)
    {
         showStatus("Key Pressed");
    }
    public void keyReleased(KeyEvent ke)
    {
        showStatus("Key Realeased");
    }
    public void keyTyped(KeyEvent ke)
    {
         msg +=ke.getKeyChar() ;
         repaint();
    }
    public void paint(Graphics g)
    {
        g.drawString(msg,100,200);
    }
}
```

❖ **Example 11: Write a program to Throw Exception Using Scanner Class**

```
import java.util.*;
class Throw3
{
    public static void main(String[] args) throws Exception
    {
        String s;
        Scanner x=new Scanner(System.in);
        System.out.println("Enter String");
        s=x.nextLine();
        if(s.equals("hello"))
        {
            System.out.print("Both string are same");
        }
        else
        {
            throw new Exception("Nomatch Exception");
        }
    }
}
```