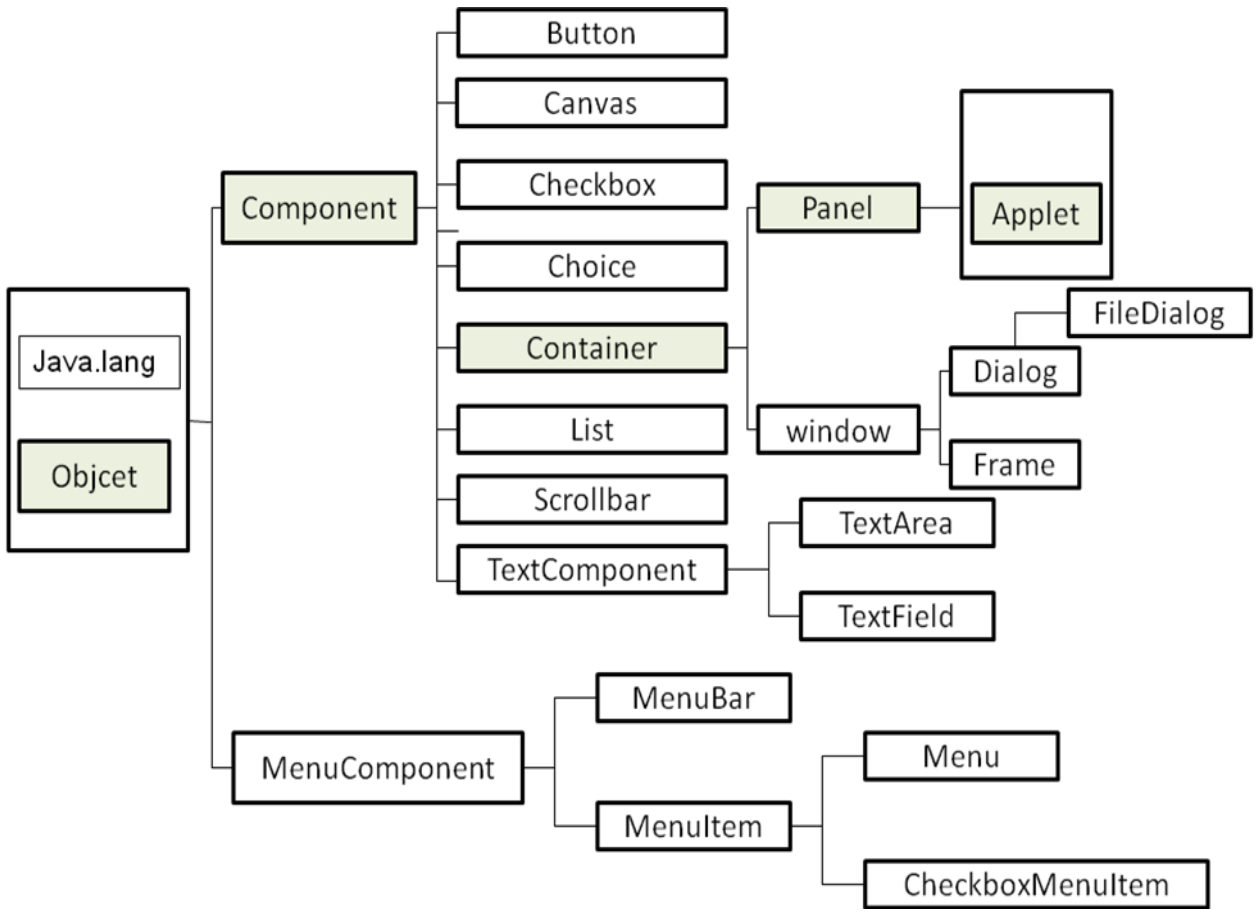
❖ Introduction to AWT:

➢ AWT class:

- **AWT means Abstract Window Toolkit. AWT** is a library of classes which provides GUI tools to develop **GUI** applications and applets.
- AWT is an *object-oriented GUI framework*.
- AWT classes are contained in java.awt package.
- It contains many classes and methods that allow you to create and manage windows and provides *machine independent interface for applications*.
- AWT is **one of the largest packages.**
- **AWT contains** classes that can be extended and their properties can be inherited, and also be abstract.
- Every **GUI** component must be a subclass of object class in **java.lang** package.
- **AWT** is used to build the graphical user interface with standard windowing elements.
- **GUI= Graphical User Interface**



➢ Component Class

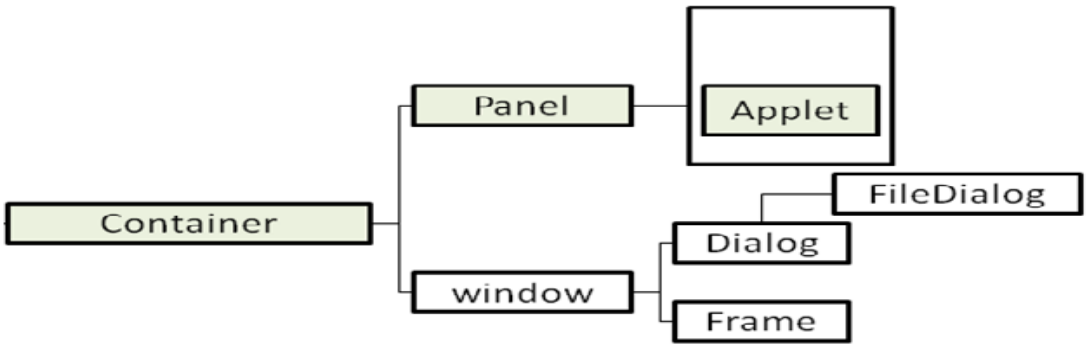
- It is the base class for all AWT and Swing components.
- It defines many basic methods inherited by all components.
- A *component* is an object having a graphical representation that can be displayed on the screen and that can interact with the user.
- Examples of components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.
- The Component class is the abstract superclass of the non menu-related Abstract Window Toolkit components.
- Class Component can also be extended directly to create a lightweight component.
- A lightweight component is a component that is not associated with a native window. On the contrary, a heavyweight component is associated with a native window. The isLightweight() method may be used to distinguish between the two kinds of the components.
- Lightweight and heavyweight components may be mixed in a single component hierarchy. However, for correct operating of such a mixed hierarchy of components, the whole hierarchy must be valid. When the hierarchy gets invalidated, like after changing the bounds of components, or adding/removing components to/from containers, the whole hierarchy must be validated afterwards by means of the Container.validate() method invoked on the top-most invalid container of the hierarchy

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

-

➢ Container Class

- To use **AWT** components in a program, you must contain them.
- The **Container** class is a *subclass* of the component class that is used to define components that have the capability to contain other components.
- It provides methods for *adding*, *retrieving*, *displaying*, *counting and removing* the components that it contains.



There are two basic types of container:

1) Window class

2) Panel class

1) Window class: this class creates popup windows separate from the main program. It has two subclasses

- **Frame**(window that have a border and a menu bar)

- **Dialog** (a special window used in applications to select a file)

2) Panel class: A container that represents a series of an existing window.

- The **applet class** is a container that is a subclass of the panel class.
- You can place components directly on applet panel or use additional panel objects to subdivide the applet into smaller sections.
- By default component are added to container in left to right and top to bottom.

Windows Fundamental

- The AWT defines windows according to a class hierarchy that adds functionality and specificity with each level. A window provides a top level window on the screen, with no borders or menu bar.
- Two most common windows are derived from
  - **Panel,** which is used by applets
  - **Frame,** which creates a standard window.
- The window class provides an encapsulation of a generic window object.
- It is sub classed by **Frame** and **Dialog** to provide capabilities needed to support application main windows and Dialog box.
- **Default Layout for window is BorderLayout**.
- A window is top level display area exists outside the browser or applet area you are working on. It has no borders, window title, or menu bat that a typical window manager might provide. **Frame** is a subclass that adds these parts borders, window title.
- **Constructor:**

public Window (Frame parent)

When parent is minimized, Window is also minimized. So we must create Frame before we create a Window.

For applet you can pass null parameter (because there is no need to access Frame),this is one way to create Dialog.

- Window class appearance method:

1. **public void pack():**resizes the window to preferred size the components contains.
2. **public void show() :** displays the window, first call to this method generates Window Event with Id WINDOW_OPENED.
3. **public void dispose():** releases the resources of the window by hiding it and removing its peer. Calling this method generates Window Event with Id WINDOW_CLOSED.
4. **public void toFront() :** brings the window to foreground of the display.

5. **public void toBack() :** puts the window in back ground of the display.
6. **public void boolean isShowing() :** returns true if the window is visible on the screen.

Window class events:
WINDOW_**DESTROY**,
WINDOW_**EXPOSE**,
WINDOW_**ICONIFY**,
WINDOW_**DEICONIFY,**
WINDOW_**MOVED**

➢ Panel class:

- Panel is the basic building block of an applet. It provides a container with no special features.
- The panel class is a subclass of Container class that is used to organize GUI components within other container objects.
- It has single constructor that takes no parameters.
- The Applet class of the java. applet package is a subclass of the Panel class.
- The default layout for a Panel object is Flow Layout.
- A panel container is not visible when it is added to an applet. It is to provide a way to organize components in a window.
- By default component are added to container in left to right and top to bottom.

```
Panel p=new Panel( );                  //creates Panel class's object
Label l=new Label("ok");
p.add( l );                            //add control to panel object
add(p);                                //add panel object to container
```

➢ Frame class:

- The frame class is used to provide the main window of an application. it may also include menu bar.
- It is a subclass of Window that supports the capabilities to specify the icon, cursor, Menu bar, and title.
- It implements MenuContainer interface so it can work with **Menu bar** object.
- **Default layout is Border Layout.**Frame provides basic building block for screen oriented applications.
- **Frame constructor:**

1. **public Frame( ):** creates a hidden window with a window title of "untitled" or an empty string.

```
Frame f1=new Frame( );
```

2. **public Frame(String title):** creates a hidden window but sets the window title to title.

```
Frame f1=new Frame("My frame");
f1.setVisible( true);          //it displays frame in output
f1.setSize( 300, 400);         //width and height of frame
```

- Frame allows you to change the mouse cursor, set an icon image, and have menus. Frame is a special type of window that looks like other high level programs in your windowing environment.
- Frame class defines 14 constants that are used to specify different types of cursors to be used within the frame.

```
public final static int DEFAULT_CURSOR
public final static int CROSSHAIR_CURSOR
public final static int TEXT_CURSOR
public final static int WAIT_CURSOR
public final static int SW_RESIZE_CURSOR
public final static int SE_RESIZE_CURSOR
public final static int NW_RESIZE_CURSOR
public final static int NE_RESIZE_CURSOR
public final static int N_RESIZE_CURSOR
public final static int S_RESIZE_CURSOR
public final static int E_RESIZE_CURSOR
public final static int W_RESIZE_CURSOR
public final static int HAND_CURSOR
public final static int MOVE_CURSOR
```

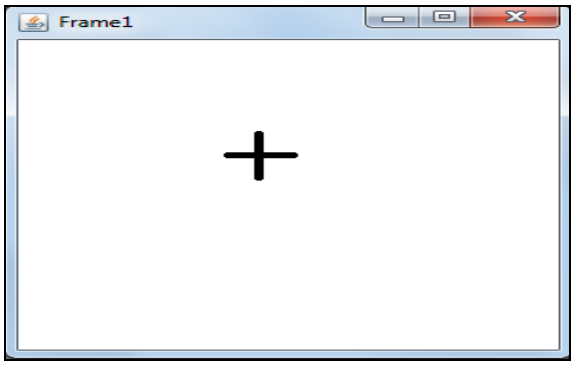
- **Frame methods:**

1. public String getTitle(): get title of frame object
2. public String setTitle(String title): set title for frame object
3. public Image getIconImage():Returns the image to be displayed as the icon for this frame.
4. public Image setIconImage(): it sets image icon to be displayed
5. public MenuBar getMenuBar():Gets the menu bar for this frame.
6. public Syncronized void setMenuBar():it sets the menu bar for this frame.
7. public Syncronized void remove(Menucomponent):Removes the specified menu bar from this frame.
8. public Syncronized void dispose():

9. public boolean isResizable():Indicates whether this frame is resizable by the user.
10. public boolean setResizable(boolean resizable):it set resizable or not.
11. public int getCursorType():it returns current cursor type

Example:

```
import java.awt.*;
import java.awt.event.*;
 public class Framedemo extends Frame
{
        Framedemo()
        {
             addWindowListener(new WindowAdapter()
               {
                        public void windowClosing(WindowEvent we)
                        {
                        System.exit(0);
                        }
                 });
}

        public static void main(String args[])
        {
            Framedemo f=new Framedemo();
            f.setVisible(true);
            f.setSize(300,300);
            f.setTitle("Frame1");
            f.setCursor(Cursor.CROSSHAIR_CURSOR);
          }
}
```



➢ Dialog class

- Dialog class is a subclass of the window class that is used to implement dialog box windows.
- A dialog box is a window that takes input from the user.

- Border layout is a default layout.
- A dialog is a popup used for user interaction; it can be modal to prevent the user from doing anything either the application before responding.
- These classes provide handful access methods, which are used to get and sets title, determine whether it is modal, and get and set the dialog box's resizable properties.
- Dialog class provides a special window that is normally used for pop up messages or input from the user.
- **Constructor:**

1. **public Dialog(Frame** parent**)**
2. **public Dialog(Frame** parent **, String** title**)**
3. **public Dialog(Frame** parent**t ,String** title **, boolean** modal**)**

- **Methods:**

1. String getTitle() :Gets the title of the dialog.
2. void hide(): to hide dialog box
3. boolean isModal() :Indicates whether the dialog is modal.
4. boolean isResizable() :Indicates whether this dialog is resizable by the user.
5. protected String paramString() :Returns a string representing the state of this dialog.
6. void setResizable(boolean resizable) :Sets whether this dialog is resizable by the user.
7. void setTitle(String title) :Sets the title of the Dialog.
8. void setVisible(boolean b) :Shows or hides this Dialog depending on the value of parameter b.
9. void show():dispays dialog
10. void toBack() :If this Window is visible, sends this Window to the back and may cause it to lose focus or activation if it is the focused or active Window.

➢ Canvas class

- The canvas component is a section of a window used primarily as a place to draw graphics or display images.
- Canvas is more similar to a container however Canvas component can't be used as a place to put components.
- Canvas class implements a GUI objects that supports drawing.
- Drawing is not implemented on the canvas itself, but on **Graphics** object provided by the canvas.
- It provides single parameter constructor and paint ( ) method.

Example:

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

```
import java.applet.*;
/*      <applet code="AppletFrame" width=400 height=60> </applet>    */
public class Appletpro extends Applet
{
   Canvas c;
   public void init()
   {    c=new Canvas();
        c.setSize(50,50);
        c.setBackground(Color.red);
        add(c);
   }
}
```

➢ AWT controls:

- Label
- TextField
- TextArea
- Buttons
- Checkboxes
- CheckboxGroup
- ChoiceList
- Scrolling lists
- Scrollbars
- Menus

1) Label

- Labels are component that are used to display text in a container which cant be edited by user.
- They are generally used to guide the user to perform correct actions on the user interface, also used for naming other components in the container.

```
Label l1=new Label ("name");
add (l1);
```

- **Label constructor**:

1) **Label( )** :creates an empty label, with its text aligned to left.
2) **Label(String):**creates label with given text, aligned left.
3) **Label(String ,int**) :creates a label with given text string and given alignment value.**(0 is Label. LEFT , 1 is Label. RIGHT ,2 is Label. CENTER**)

Methods:

1. getText() :it returns a string containing this label's text.
2. setText(String) : it set a new text of label

3. GetAlignment( ):it returns an integer value representing the alignment of label (0 is Label.LEFT , 1 is Label. RIGHT ,2 is Label. CENTER)
4. SetAlignment(int) : changes the alignment of this label to the given integer value (0,1 or 2)

Example:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="Labels" width=400 height=400></applet>*/
public class Labels extends Applet
{

  public void init() {
    String s = "This is a very long label";
    Label l1 = new Label(s, Label.LEFT);
    add(l1);
    Label l2 = new Label(s, Label.CENTER);
    add(l2);
    Label l3 = new Label(s, Label.RIGHT);
    add(l3);
  }
}
```

2) TextFields

- TextFields is used for inserting a single line of text.
- It creates an empty text field with an unspecified number of columns.

```
TextField t1=new TextField (8);
 add(t);
```

- **TextFields Constructors**:

1) **TextField( ):**creates an empty textfield is 0 character wide.
2) **TextField(int):**creates empty text field. Integer argument defines no. Of characters to display.
3) **TextField(String):**creates a text fields initialized with given string.
4) **TextField(String ,int)**:creates a text field some number of characters wide.

Methods:

- **getText( )** **:**Returns text contained in text field
- **setText (String)** :Puts the text in text fields
- **getColumns( )** **:**Returns width of this text field
- **select(int, int)** **:**Selects text between two integer positions(starts from 0)
- **selectAll( )** **:**Selects all text in the field

- **isEditable ( ) :**Returns true or false based on whether text is editable.
- **setEditable(boolean**): True enables text to be edited, false freezes the text
- **getEchoChar( ) :**Returns character used for making input.

3) TextArea

- Creates an empty text area with an unspecified number of rows and columns.
- The number of rows and columns are specified as parameters to the constructor
- It provides the capability to set the text to read-only or edit mode.

```
TextArea t=new TextArea(5,20);
Add(t);
```

- **TextArea constructor**:

1. **TextArea():**creates an empty text area 0 rows long and 0 characters wide.
1. **TextArea(int, int):**creates an empty text area with given number of rows and columns
2. **TextArea(String):**creates a text area displaying the given string.
3. **TextArea(String, int, int):**creates a text area displaying the given string and with given dimensions

TextArea t=new **TextArea**("hi \n hello",10,25);

Methods:

1. **getColumns( ) :**returns the width of the text area, in characters or columns
2. **getRows( ):**returns the number of rows in the text area
3. **insertText(String,int) :**inserts the string at the given position in the text
4. **replaceText(String,int,int):**replaces the text between the given integer positions with the new string

4) Button

- Button component is used to trigger events in GUI.
- It is rectangular button that can be clicked with a mouse.
- Buttons are easy to manage and make the interface presentable.
- **Button constructor:**
    1. Button( ): creates an empty button with no label
    2. Button(String) :creates a button with the given string object as a label.

    Button b=new Button("ok");
    Add(b);
- Once you have button object ,you can get the value of the button's label by getLabel( )method and set the label using setLabel( ).

We can add button in panel using following code:

```
Panel  p=new Panel ();
Button b=new Button("ok");
p.add(b);
```

5) Checkboxes

- A check box is a used to control in an application or a web page.
- A check box has two parts ,a **lable** and a **state**
- Label text is text that represents the caption of the control
- The state is a boolean value that represents the status of check boxes whether checked or unchecked.By default the state is false, means checkbox is unchecked.
- **Checkbox constructor:**

  1. Checkbox( ) :creates an empty checkbox ,unselected

  2. Checkbox(String) :creates a checkbox with the given string as a label.

  3. Checkbox(String ,null, boolean) :creates a checkbox that is either selected or unselected based on whether the boolean argument is true or false

Syntax:

```
Checkbox ch1=new Checkbox("pen");
add(ch1);
```

Methods:

1. **getLabel( ) :** returns a string containing checkbox's label
2. **setLable(String):** changes the text of the check box's label
3. **getState() :**returns true or false, based on whether the check box is selected
4. **setState(boolean):** changes the check box's state to selected (true) ot unselected (false)

6) CheckBoxGroup

- **CheckBoxGroup** class is used with the **Checkbox** class to implement radio buttons.
- All **Checkbox** objects associated with **CheckBoxGroup** object are treated as a single set of radio buttons.
- Only one button in group may be set or on at a given point in time.
- **CheckboxGroup constructor:** it provides a single parameter less constructor.
- We can get and set CheckboxGroup object.

```
Checkbox ch1=new Checkbox("male");
ch1.setCheckboxGroup(cbg);
ch1.setState(false);
 add(ch1);
```

7) ChoiceList

- Choice class is used to implement **pull-down lists** that can be placed in the main area of a window.
- These lists are known as **option menus** or a **pop-up menu** of choices.
- It is a pop up list of strings from which ha single string can be selected. It enables selection of one Item at a time.

```
Choice  c=new Choice();
c.addItem("bhuj");
c.addItem("rajkot");
add(c);
```

Methods:

1. **getItem(int):** returns the string item at given position (items inside a choice begin at 0.just like arrays)
2. **countItems( ):**returns the number of items in the menu
3. **select(int) :** select the item at the given position
4. **Select(string):**select the item with the given string
5. **getSelectedIndex( ):** returns the index position of the item selected
6. **getSelectedItem( ):** returns the currently selected item as a string

8) Scrolling List

- The List class implements single and multiple selection list GUI controls.
- A list box is similar control to a choice list, however unlike choice menu; a list control allows the user to select multiple values in the list as the same time.
- One or more string can be selected at same time.
- **Scrolling List constructor**:

1) List( ) :creates an empty scrolling list that enables only one selection at a time.
2) List(int, boolean):creates a scrolling list with the given number of visible lines on the screen. integer argument defines no. Of element to be displayed on screen and boolean specifies whether list enables multiple choice or not.

Methods:

1. void add(String item) :Adds the specified item to the end of scrolling list.
2. void add(String item, int index) :Adds the specified item to the the scrolling list at the position indicated by the index.
3. void addActionListener(ActionListener l) :Adds the specified action listener to receive action events from this list.

4. void addItem(String item)
5. void addItem(String item, int index)
6. boolean allowsMultipleSelections() :set to select multiple selections
7. int countItems() : count no. of items in list
8. String getItem(int index) :Gets the item associated with the specified index.
9. int getItemCount() :Gets the number of items in the list.
10. ItemListener[] getItemListeners() :Returns an array of all the item listeners registered on this list.
11. String[] getItems() :Gets the items in the list.
12. int getSelectedIndex() :Gets the index of the selected item on the list,
13. int[] getSelectedIndexes() :Gets the selected indexes on the list.
14. String getSelectedItem() :Gets the selected item on this scrolling list.
15. String[] getSelectedItems() :Gets the selected items on this scrolling list.
16. boolean isSelected(int index) :checks item is selected or not
17. void remove(int position) :Removes the item at the specified position from this scrolling list.
18. void remove(String item) :Removes the first occurrence of an item from the list.
19. void removeAll() :Removes all items from this list.
20. void select(int index) :Selects the item at the specified index in the scrolling list.

9) Scroll Bars

- Scrollbar is used to implement vertical and horizontal scrollbars.
- Scrollbar component is up-down or left-right slider that can be used to set a numeric value.You can use scrollbar by clicking mouse or an arrow or by grabbing the box on the slider.
- **Scrollbar constructor:**

1. **scrollbar()**:creates scrollbar with its initial maximum and minimum value 0,in vertical .
2. **scrollbar(int):**creates scrollbar with its initial maximum and minimum value 0,the argument can be set to scrollbar. HORIZONTAL or scrollbar. VERTICAL.
3. **Scrollbar(int orientation, int intial, int overallwidth, int min, int max):**

First argument: orientation of the scroll bat scrollbar. HORIZONTAL or scrollbar. VERTICAL.

Second argument: initial value of scroll bar ,which should be a value between scrollbar 's minimum & maximum value.

Third argument: overall width (or height)

fourth and fifth: scrollbar. Minimum and maximum value for the scroll bar.

Methods:

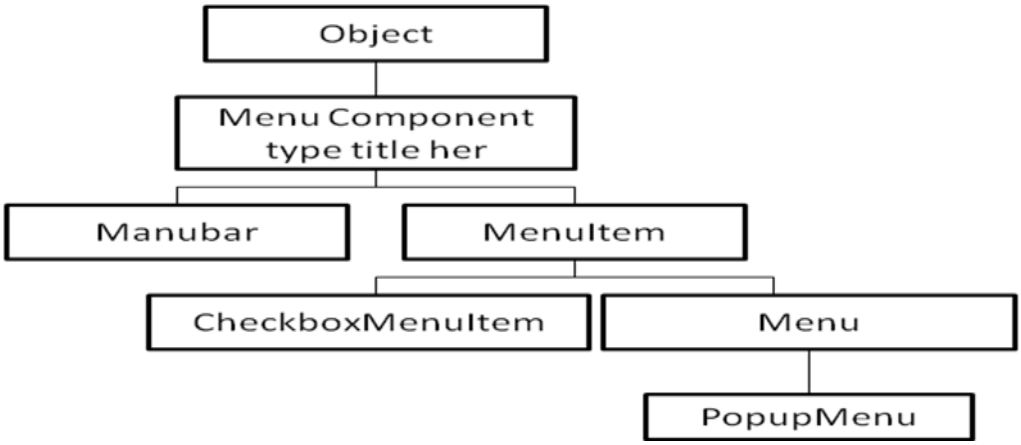
1. getMinimum( ): Gets the minimum value of this scroll bar
2. getMaximum( ): Gets the maximum value of this scroll bar
3. getOrientation( ) : Returns the orientation of this scroll bar, 0 for Horizontal 1 for vertical
4. getValue():Gets the current value of this scroll bar
5. setValue() : setts the new value of this scroll bar
6. setLineIncrement() : sets value of the scrollbar is changed when incrementing in line mode
7. getLineIncrement():gets value of the scrollbar is changed when incrementing in line mode
8. setPageIncrement(int c): sets value of the scrollbar when incrementing in page modedefault is 10
9. getPageIncrement():sets value of the scrollbar when incrementing in page mode.

10) Menus

- A menu is component of AWT but it is different from other components because it can't be added to ordinary container and laid out by layout manager.
- Menu can be added only to a menu container. Top level window can have menu bar which display list of top level menu choices and each choice is associated with a dropdown menu.

MenuBar

- A MenuBar component is a horizontal menu in which contains on more Menu objects.
- It can only be added to Frame object.
- It forms the root of all menu trees.



- A Frame can display only one MenuBar at a time.
- The Menu Bar does not support any listener.

MenuBar Constructor:

- **MenuBar ( ):** to create default MenuBar.

Menu:

- A menu component provides a basic pull down menu.
- It can be added either to a menu bar or to another Menu.
- Menus are used to display and control Menu items.

Menu Constructor:

1. **Menu () :** to create default menu.
2. **Menu (String str):** str specifies name of the menu selection
3. **Menu(String str, Boolean flag**):flag shows popup menu if set true it can be removed and allowed to float free.

MenuItem

- MenuItem component are text leaf nodes of a menu tree.
- MenuItem are added to a Menu.
- An actionListener can be added to a MenuItem Object.

MenuItem constructor:

1. **MenuItem () :**create s a default constructor
2. **MenuItem (String str) :** str is the name shown in the menu.
3. **MenuItem (String str,MenuShortcut key):**key is shortcut key for that MenuItem.

CheckboxMenuItem

- It is checkable menu item, which provides selection (on or off) listed in menus.
- It can be controlled by the ItemListener interface.

CheckboxMenuItem constructor:

1. CheckboxMenuItem() :

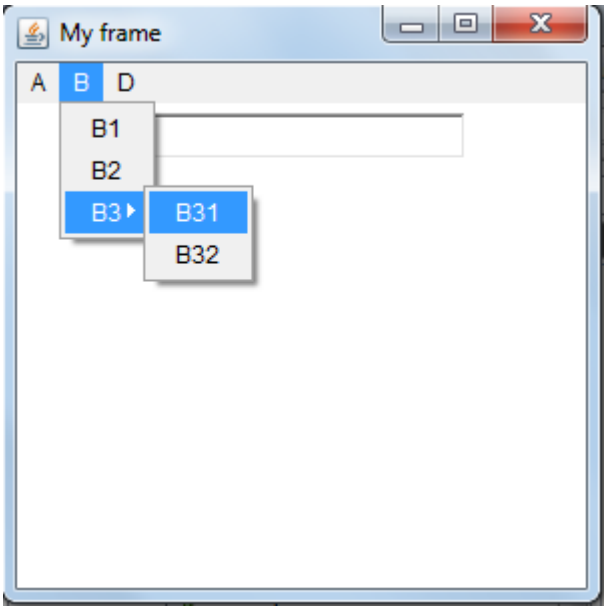
2. CheckboxMenuItem(String str):
3. CheckboxMenuItem(String str, boolean flag):

Example:

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

public class Prac9 extends Frame implements ActionListener,ItemListener
{
   TextField tf=new TextField(20);

public static void main(String args[])
{
   Prac9 mn=new Prac9("My frame");
   mn.setVisible(true);
   mn.setSize(300,300);
   mn.setTitle("My Frame Menu");
   }
Prac9(String title)
{
   super(title);
   FlowLayout f=new FlowLayout(FlowLayout.CENTER);
   setLayout(f);

   addWindowListener(new WindowAdapter()   //to close frame window
      {
            public void windowClosing(WindowEvent e)
            {
               System.exit(0);
            }
      });
   MenuBar mb=new MenuBar();              //creates menubar
   setMenuBar(mb);

   Menu a=new Menu("A");                  //creates menu and add it to menubar
   mb.add(a);
```

```
MenuItem a1=new MenuItem("A1");        //creates menuitem to add in menu
a1.addActionListener(this);
a.add(a1);
MenuItem a2=new MenuItem("A2");
a2.addActionListener(this);
a.add(a2);

Menu b=new Menu("B");
mb.add(b);
MenuItem b1=new MenuItem("B1");
b1.addActionListener(this);
b.add(b1);
MenuItem b2=new MenuItem("B2");
b2.addActionListener(this);
b.add(b2);

Menu b3=new Menu("B3");            //creates submenu in menu B
b.add(b3);
MenuItem b31=new MenuItem("B31");
b31.addActionListener(this);
b3.add(b31);

MenuItem b32=new MenuItem("B32");
b32.addActionListener(this);
b3.add(b32);

Menu d=new Menu("D");
mb.add(d);
CheckboxMenuItem d1=new CheckboxMenuItem("D1");
d1.addItemListener(this);
d.add(d1);

CheckboxMenuItem d2=new CheckboxMenuItem("D2");
d2.addItemListener(this);
d.add(d2);

add(tf);
```

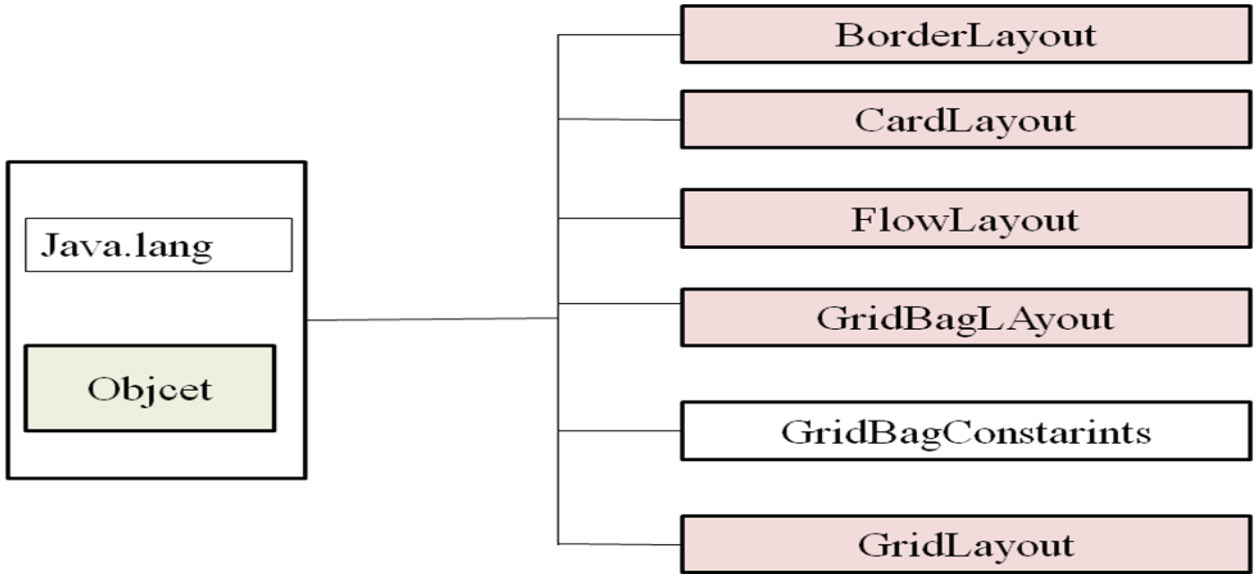
```
}
   public void actionPerformed(ActionEvent e)
   {
      tf.setText("Action event:"+e.getActionCommand()+"\n");
   }

  public void itemStateChanged(ItemEvent ie)
   {
      CheckboxMenuItem chm=(CheckboxMenuItem)ie.getSource();
      tf.setText("Itemevent:"+chm.getLabel()+"\n");
   }

}
```



➢ Layout Manager

- There's no guarantee that a textbox or any component will be the same size on each platform.
- Sun Microsystems has created a Layout Manager interface that defines methods to reformat the screen based on the current layout and component sizes.
- **Layout manager try to give a consistent and reasonable appearance regardless of platform, the screen size, or actions the user might take.**
- It determines that how awt components are dynamically arranged on the screen.
- Layout manager is an object that is used to organize components in container.
- Every container has default layout manager.

- As soon as you create container java automatically creates and assigns layout manager to it.



1. Flow Layout (default Layout)

- It is default layout for applets and Panels.
- It places controls in the order in which **they are added, linearly, from left to right and from top to bottom in horizontal and vertical rows.**
- We can align components to left, right or center.
- Components are normally centered in applet.
- When the layout manager reaches the right border of the container (means row is full then), it positions the components in the next row.
- **If you resize an applet, the components flow will change based upon the new width and height.**

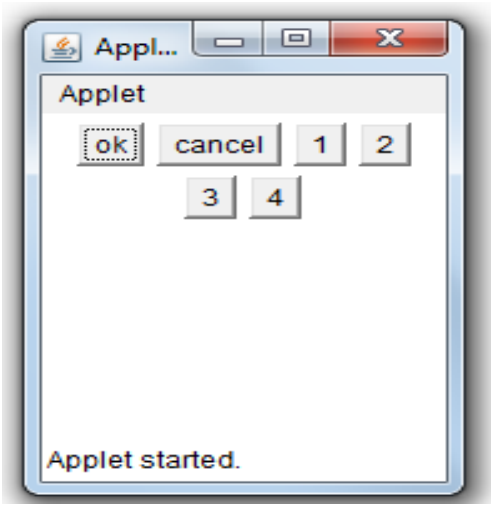
To create flow Layout :

```
FlowLayout f1=new FlowLayout();
FlowLayout f2=new FlowLayout(FlowLayout.LEFT);
FlowLayout f3=new FlowLayout(FlowLayout.LEFT,10,30);
setLayout(f1)
```

Example:

```
import java.applet.*;
import java.awt.*;
/*      <applet code=Flowtest.class width=150 height=400>    </applet> */
public class Flowtest extends Applet
{
        Button bt1=new Button("ok");
        Button bt2=new Button("cancel");
        Button bt3=new Button("1");
```

```
        Button bt4=new Button("2");
        Button bt5=new Button("3");
        Button bt6=new Button("4");

        FlowLayout f1=new FlowLayout(FlowLayout.LEFT);
        FlowLayout f2=new FlowLayout(FlowLayout.RIGHT);
        FlowLayout f3=new FlowLayout(FlowLayout.CENTER);
        public void init()
        {
        setLayout(f3);
        add(bt1);
        add(bt2);
        add(bt3);
        add(bt4);
        add(bt5);
        add(bt6);
        }
}
```

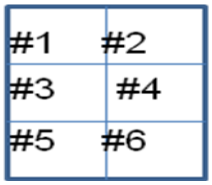


2. Grid Layout:

- The Grid layout class puts each component into a place on a grid that is equal in size to all the other places.
- The grid is given specific dimensions when created; Components are added to the grid in order, starting **with upper-left corner to right**.
- Components are given equal amounts of space in the container.
- Grid layout is widely used for arranging components in rows and columns.
- However unlike, Flow Layout, here underlying components are resized to fill the row-column area. if possible.

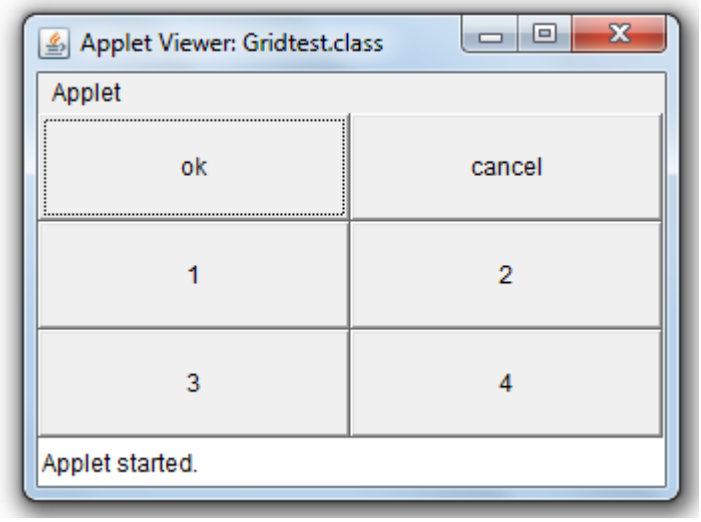
- Grid Layout can reposition objects after adding or removing components. Whenever area is resized, components are also resized.

GridLayout b1=new GridLayout(4,1); // 4 rows and 1 column
setLayout(b1)



Example :

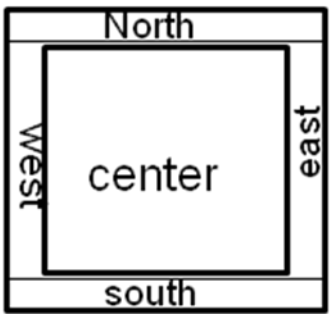
```
import java.applet.*;
import java.awt.*;
/*      <applet code=Gridtest.class width=100 height=100>       </applet>     */
public class Gridtest extends Applet
{
        Button bt1=new Button("ok");
        Button bt2=new Button("cancel");
        Button bt3=new Button("1");
        Button bt4=new Button("2");
        Button bt5=new Button("3");
        Button bt6=new Button("4");
        GridLayout f1=new GridLayout(3,2);
        public void init()
        {
        setLayout(f1);
        add(bt1);
        add(bt2);
        add(bt3);
        add(bt4);
        add(bt5);
        add(bt6);
        }
}
```



3. Border Layout

- Border layout is the default layout manager for all window, dialog and frame class.
- In Border layout ,components are added to the edges of the container ,the center area is allotted all the space that's left over.
- Border layout ,provides five areas to hold component:
  Four border→ north, south, east, west
  Remaining space: center
- When you add a component to the layout you must
  Specify which area to place it in.
- The order in which components are added to the screen is not important. Although you have only one component in each area.

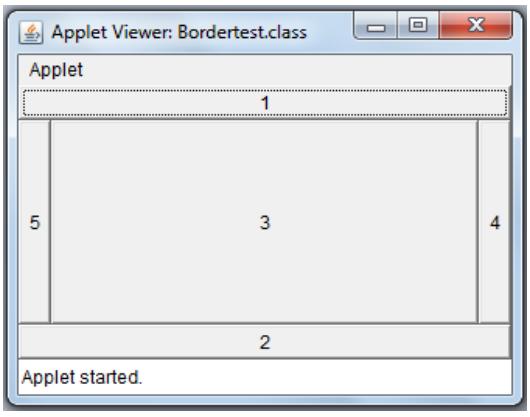
```
BorderLayout b1=new BorderLayout ();
setLayout (b1);
add (bt1,BorderLayout.NORTH);
add(bt2,BorderLayout.SOUTH);
```



Example:

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

```
<applet code=Bordertest.class width=100 height=100>
</applet>
*/
public class Bordertest extends Applet
{
Button bt1=new Button("1");
Button bt2=new Button("2");
Button bt3=new Button("3");
Button bt4=new Button("4");
Button bt5=new Button("5");
Button bt6=new Button("6");
BorderLayout b1=new BorderLayout( );
public void init()
{
setLayout(b1);
add(bt1,BorderLayout.NORTH);
add(bt2,BorderLayout.SOUTH);
add(bt3,BorderLayout.CENTER);
add(bt4,BorderLayout.EAST);
add(bt5,BorderLayout.WEST);
//add(bt6,BorderLayout.WEST);
}
}
```



4. Card Layout

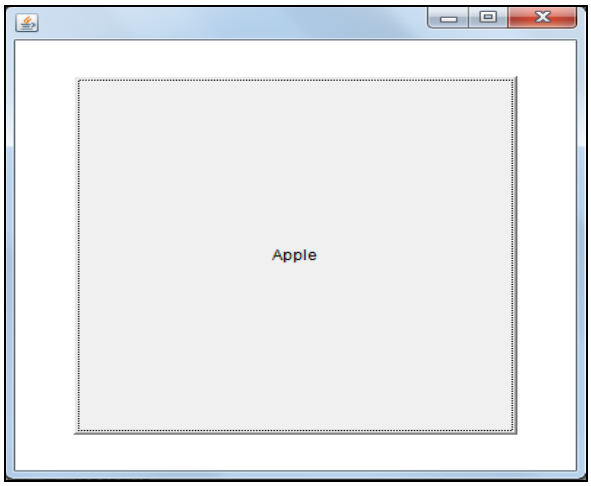
- Card layout class is a special type of layout organizer.
- *A card layout manages several components, displaying one of them at a time and hiding the rest.*

- All the components are given same size, Usually Card Layout manages a group of panels and each panel contains several component of its own.
- It doesn't display several panels concurrently but it creates a stack of panels that can then be displayed one at a time, as stack of cards in solitaire game.
- Card Layout allows you to assign names to the components it is managing and lets you jump to a component by name.
- Card Layout class has its own methods that are used to control which panel is displayed. Clicking on applet area causes the Layout manager to goes the next panel.

Example :

```
import java.awt.*;
import java.awt.event.*;
 public class CardLayoutExample extends Frame implements ActionListener
{
        CardLayout card;
        Button b1,b2,b3;
        Panel p;
           CardLayoutExample()
           {
              addWindowListener(new WindowAdapter()
                {
                   public void windowClosing(WindowEvent e)
                  {
                     System.exit(0);
                  }
        });
       p=new Panel();
      card=new CardLayout(40,30);  //create CardLayout object with 40 hor space and 30 ver space
      p.setLayout(card);

      b1=new Button("Apple");
      b2=new Button("Banana");
      b3=new Button("Grape");
      b1.addActionListener(this);
      b2.addActionListener(this);
      b3.addActionListener(this);

      p.add("a",b1);
      p.add("b",b2);
```

```
        p.add("c",b3);
        add(p);
    }
    public void actionPerformed(ActionEvent e)
        {
    card.next(p);
            }
   public static void main(String[] args)
        {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        }
}
```



5. GridBag Layout

- GridBagLayout class provides a grid for components like GridLayout, but allows a single component element to occupy multiple cells of the grid.
- Each GridBagLayout uses a rectangular grid of cells, just like a GridLayout.
- Cells are determined and shaped by the component placed in them rather than components being shaped to fir the cells.
- Cells in grid are not required to take up same amount if space and components can be aligned in different ways in each grid cell.
- GridBagLayout constructor is trivial, with no arguments.
- **GridBagLayout gb=new GridBagLayout();**
- Unlike **GridLayout ( )** constructor, this constructor does not specify the number of rows or columns.

- **Rows and columns are added as required (starts from 0.)**
- You will need access to the **GridBagLayout** object later in the applet when you add components to the container.

GridBagConstraints objects

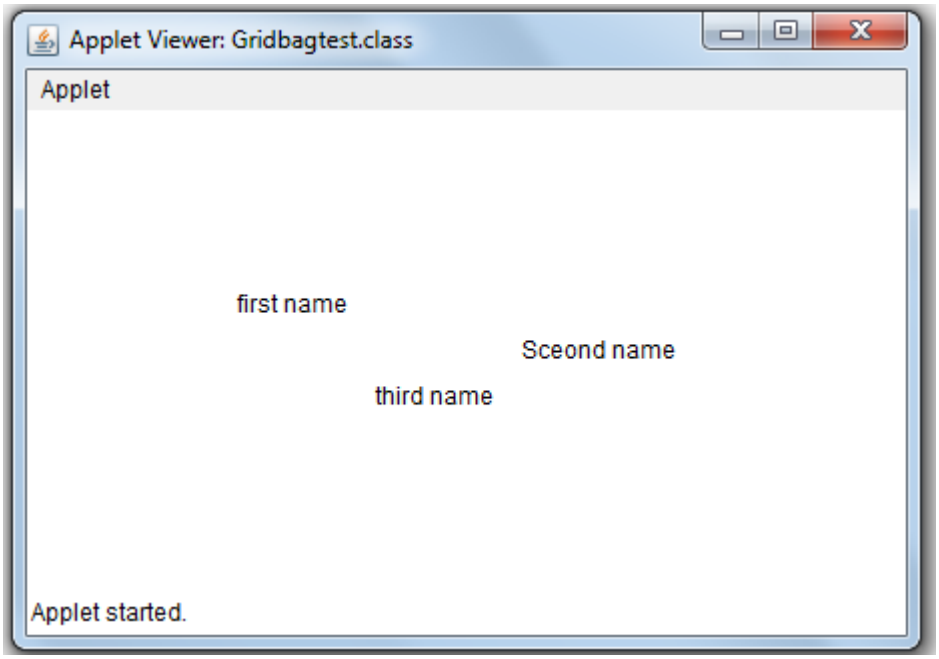
- GridBagConstraints object specifies the location and area of the components display area within the container, and how they are laid out inside its display area.

GridBagConstraints gbc=new **GridBagConstraints();**

Example:

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

/*
<applet code=Gridbagtest.class width=100 height=100>
</applet>
*/
public class Gridbagtest extends Applet
{
public void init()
{
        GridBagLayout g1=new  GridBagLayout();
        GridBagConstraints gb=new GridBagConstraints();
        setLayout(g1);

        Label l1=new Label("first name");
        Label l2=new Label("Sceond name");
        Label l3=new Label("third name");

        gb.gridx=1;
        gb.gridy=1;
        gb.gridwidth=2;

        g1.setConstraints(l1,gb);
        add(l1);

        gb.gridx=5;
        gb.gridy=3;
        gb.gridwidth=2;

        g1.setConstraints(l2,gb);
```

```
        add(l2);

        gb.gridx=3;
        gb.gridy=5;
        gb.gridwidth=2;

        g1.setConstraints(l3,gb);
        add(l3);

    }


    }
```



➢ Swing

- **Swing** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Swing is a set of classes that provide more powerful and flexible components than AWT.
- Swing provides familiar components like buttons, checkbox and label and also adds new components like tabbed pane, scroll panes, tree and tables.
- In swing button may have both an image and a text string associated with it. the image can be changed as the state of the button changes.
- The swing related classes are contained in **javax.swing** and its sub packages.
- The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Difference between AWT and Swing

No.	AWT	Swing
1	AWT stands for Abstract windows toolkit.	Swing is also called as JFC's (Java Foundation classes).
2	AWT components are called Heavyweight component.	Swings are called light weight component because swing components sits on the top of AWT components and do the work.
3	AWT components require java.awt package.	Swing components require javax.swing package.
4	AWT components are platform dependent.	Swing components are made in purely java and they are platform independent.
5	This feature is not supported in AWT.	We can have different look and feel in Swing.
6	This feature is not available in AWT.	Swing components are called "lightweight" because they do not require a native OS object to implement their functionality components like JButton, JTextArea, etc., are lightweight because they do not have an OS peer.
7	AWT is a thin layer of code on top of the OS.	Swing is much larger. Swing also has very much richer functionality.
8	Using AWT, you have to implement a lot of things yourself.	Swing has them built in.
9	With AWT, you have 21 "peers" (one for each control and one for the dialog itself). A "peer" is a widget provided by the operating system, such as a button object or an entry field object.	With Swing, you would have only one peer, the operating system's window object. All of the buttons, entry fields, etc. are drawn by the Swing package on the drawing surface provided by the window object.

- The swing component classes are as follow:
  1. JApplet
  2. JLabel
  3. JTextField
  4. JButton
  5. JCheckBox

6. JRadioButton
7. JComboBox
8. JtabbedPane
9. JMenus

1. JApplet :

- Fundamental of Swing is **JApplet** class, which extends Applet.
- Applets that use Swing must be subclasses of JApplet,do.
- JApplet is rich with functionality .It supports various "panes",such as **content pane,glass pane and root pane.**
- When adding a component to an instance of JApplet,do not invoke the add( ) method of the applet.but call add( )for content pane of the JApplet object.
- We can obtain content pane using **getContentPane( )** method.
- The **add( )** method of container can be used to add a component to content pane.
  void add (component) //component which you want to add

4. JLabel:
5.

- Swing labels are instances of JLabel class. which extends JComponenets.
- It can display text and or icon.
- **Constructor of JLabel**
  1. **JLabel(** Icon i)
  2. **JLabel**(String s)
  3. **JLabel(**String s, Icon I,int align)
     Align is either LEFT,RIGHT,CENTER,LEADING or TRAILING.
- **Methods:**
  1. **Icon getIcon( )**
  2. **String getText( )**
  3. **void setIcon( )**
  4. **void setText( )**

3. JTextField:

- Swing text field is encapsulated by JTextComponent class, which extends JComponent.
- It provides functionality that is common to Swing text Components. One of its sub class is JTextField,which allows you to edit one line of text.
- **Constructor**
  1. JTextField( )
  2. JTextField(int cols)

3. JTextField(String s,int cols)
4. JTextField(String s)

4. JButton:

- Swing buttons provide features that are not found in the Button class defined by the AWT.
- Swing buttons are subclasses of AbstractButton class which extends JComponent.
- AbstractButton is a super class for push buttons, checkboxes and radio buttons. It contains many methods that allow you to control the behaviour of buttons, checkboxes and radio buttons.
- Concrete subclasses of AbstractButton generate action events when they are pressed. Method used for even handling are,

  void addActionListener(ActionListener l)
  void removeActionListener(ActionListener al)
- Methods for Button

1) public void setText(String s): is used to set specified text on button.

2) public String getText(): is used to return the text of the button.

3) public void setEnabled(boolean b): is used to enable or disable the button.

4) public void setIcon(Icon b): is used to set the specified Icon on the button.

5) public Icon getIcon(): is used to get the Icon of the button.

6) public void setMnemonic(int a): is used to set the mnemonic on the button.

7) public void addActionListener(ActionListener a): is used to add the action listener to this object.

- **JButton** class provides the functionality of a push button.JButton allows an icon,a string or both to be associated with the push button.
- Constructor for JButton:
  1. JButton(Icon i)
  2. JButton(String s)
  3. JButton(String s,Icon i) //s and i are the string and icon used for the button.

Example:

```
import javax.swing.*;
public class ImageButton
{
ImageButton()
{
   JFrame f=new JFrame();
```

```
JButton b=new JButton("Click");
b.setBounds(130,100,100, 40);

f.add(b);
f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args)
{
  new ImageButton();
}
}
```

5. JCheckBox

- **JCheckBox** class which provides the functionality of a check box, is a subclass of AbstractButton class.Its immediate superclass is JToggleButton.
- It supports two states true or false.
- We can associate an icon, string, or the state with the checkbox.
- **JCheckBox constructors:**
    1. JCheckBox(Icon c)
    2. JCheckBox(Icon c,boolean state)
    3. JCheckBox(String s)
    4. JCheckBox(String s,boolean state)
    5. JCheckBox(String s,Icon i)
    6. JCheckBox(String s,Icon c,boolean state)

  Here, i is the icon for the button.s is text to be displayed.If state is true,the checkbox is initially selected else not.
- The state of the check box can be changed via following method:

  void setSelected(boolean state)
- When a check box is selected or deselected ,an item event is generated.this is handled by **itemStateChanged( ).**
- The getItem( ) method is used to get JCheckBox object that generated the event.
- The getText( ) method gets the text for that check box and text inside the text field.
- To select only one you have to create a object of ButtonGroup class and add all components in the object.

Example :

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

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

class JCheckBoxDemo extends JFrame implements ItemListener
{
        JTextField tf;
        JCheckBox jch1,jch2;
        JCheckBoxDemo()
        {
                super("Check Box Demo");
                setLayout(new FlowLayout());
                tf = new JTextField(15);
                add(tf);
                jch1 = new JCheckBox("apple");
                add(jch1);
                jch1.addItemListener(this);
                jch2 = new JCheckBox("banana");
                add(jch2);
                jch2.addItemListener(this);
                setVisible(true);
                setSize(300,300);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
                public void itemStateChanged(ItemEvent ie)
                {
                JCheckBox jc=(JCheckBox)ie.getItemSelectable();
                tf.setText(jc.getText()+" is selected");
                }
        public static void main(String args[])
        {
                new JCheckBoxDemo();
        }
}
```

6. JRadioButton

- Radio buttons are supported by the **JRadioButton** class,which is a concrete implementations of AbstractButton.JRadioButton's immediate superclass is JToggleButton,which provides support for two state buttons.
- Radio buttons must be configured into a group.Only one of the buttons in that group can be selected at any time.

- The **ButtonGroup** class is instantiated to create a button group.Its default constructor is invoked for this purpose.Elements are then added to the button group via **add( )** method.

  void add(AbstractButton ab) //ab is reference to the button to be added to group.

- **Constructor for JRadioButton**
  1. JRadioButton (Icon c)
  2. JRadioButton (Icon c,boolean state)
  3. JRadioButton (String s)
  4. JRadioButton (String s,boolean state)
  5. JRadioButton (String s,Icon i)
  6. JRadioButton (String s,Icon c,boolean state)

Example:

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

class JRadioButtonDemo extends JFrame implements ItemListener
{
	JTextField tf;
	JRadioButton f,m;
	ButtonGroup group;
	JRadioButtonDemo()
	{
		super("Radio Button Demo");
		setLayout(new FlowLayout());
		tf = new JTextField(15);
		add(tf);
		f = new JRadioButton("Female");
		add(f);
		f.addItemListener(this);
		m = new JRadioButton("Male");
		add(m);
		m.addItemListener(this);
		group = new ButtonGroup();
		group.add(f);
		group.add(m);
		setVisible(true);
		setSize(300,300);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
```

```
                public void itemStateChanged(ItemEvent ie)
                {
                if(f.isSelected())
                {
                    tf.setText(""+f.getActionCommand());
                }
                else
                {
                    tf.setText(""+m.getActionCommand());
                }

                }

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

7. JComboBox

- Swing provides a combo box(a combination of a text field and a drop down list) through the **JComboBox** class, which extends **JComponent**.
- A combo box normally displays one entry. However, it can also display a drop-down list that allows a user to select a different entry.
- You can also type your selection into the text field.
- **Constructor of JcomboBox**
    1. JComboBox**( )**
    2. JComboBox(Vector v) //v is vector that initializes the combo box.
- Items are added to the list of choices via the **addItem( )**method,

    void addItem(Object obj) //obj is object to be added to the combo box

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

class JComboBoxDemo extends JFrame implements ItemListener
{
    JComboBox cb;
   JLabel jl;
    String lang[] = {"JAVA","DOTNET","VB","PHP","C++"};
    JComboBoxDemo()
    {
```

```
            super("Combo Box Demo");
            setLayout(new FlowLayout());
        jl=new JLabel("            ");
        add(jl);
            cb = new JComboBox(lang);
            add(cb);
            cb.addItemListener(this);
            setVisible(true);
            setSize(300,300);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void itemStateChanged(ItemEvent ie)
    {
            jl.setText("you have selected :"+cb.getItemAt(cb.getSelectedIndex()));
    }
    public static void main(String args[])
    {
            new JComboBoxDemo();
    }
}
```

8. JtabbedPane

- A tabbed pane is a component that appears as a group of folders in a file cabinet. Each folder has a title. When a user selects a folder, its content becomes visible. Only one of the folders may be selected at a time.
- Tabbed panes are commonly used for setting configuration options.Tabbed panes are encapsulated by the JtabbedPane class, extends JComponent.
- We will use its default constructor. Tabs are defined via the following method.

  void **addTab**(String str, Component comp)

  str is title for the tab comp is the component that should added to the tab. Typically, a **JPanel** or a subclass of it is added.

9 .JMenu:

- We can create a Menu Bar which contains several menus.
- Each menu can have several menu items. The separator can also be used to separate out these menus.
- JMenuBar :- This class creates a menu bar.
- JMenu(String s) :- This class creates several menus. The string s denotes the name of the menus.
- JMenuItem(String s) :- The menu items are the parts of menus. The string s denotes the name of the menu item.

- setMnemonic(char c) :- The character which is passed to it as an argument becomes the mnemonic key. Hence using alt+c you can select that particular menu.
- JSeparator :- This is the constructor of the class JSeparator which adds separating line between the menu items.
- setMenuBar :- This method is used to set menu bar to a specific frame.

Example :

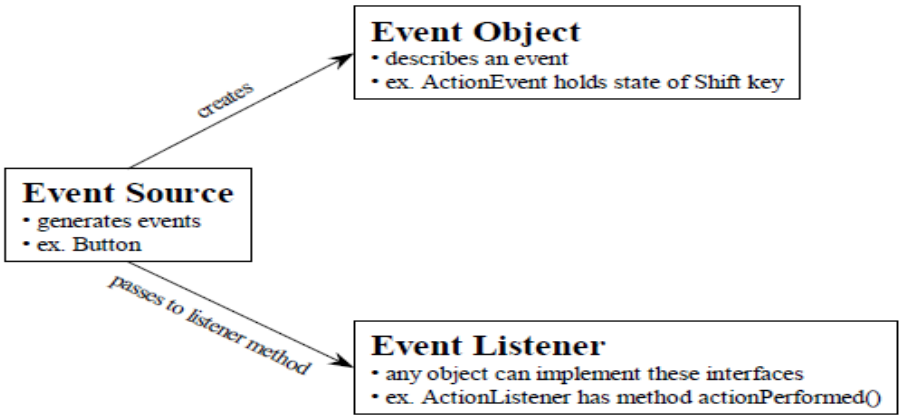
```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

class Menu extends JFrame
{
	JMenuBar menubar;
	JMenu file,color;
	JMenuItem new1,open,save,close;
	JMenuItem red,blue,green;
	JButton btnNew,btnSave,btnExit;
	Menu()
	{
		super("Menu Sample Demo");
		setLayout(new FlowLayout());
		ImageIcon fileicon = new ImageIcon("images/New.png");
		ImageIcon saveicon = new ImageIcon("images/save.png");
		ImageIcon closeicon = new ImageIcon("images/exit.png");
		btnNew = new JButton(fileicon);
		btnSave = new JButton(saveicon);
		btnExit = new JButton(closeicon);
		menubar = new JMenuBar();
		file = new JMenu("File");
		file.setMnemonic(KeyEvent.VK_F);
		add(file);
		new1 = new JMenuItem("New",fileicon);
		add(new1);
		file.add(new1);

		save = new JMenuItem("Save",saveicon);
		add(save);
		file.add(save);
		close = new JMenuItem("Close",closeicon);
		close.setMnemonic(KeyEvent.VK_X);
		add(close);
		file.add(close);
```

```
                close.addActionListener(new ActionListener()
                {
                        public void actionPerformed(ActionEvent ae)
                        {
                                System.exit(0);
                        }
                });

                color = new JMenu("Color");
                add(color);
                red = new JMenuItem("Red");
                add(red);
                color.add(red);
                menubar.add(file);
                menubar.add(color);
                setJMenuBar(menubar);
                setVisible(true);
                setSize(300,300);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

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

➢ Event Handling:

The delegation event model

- The modern approach to handling event is based on delegation event model, which defines standard and consistent mechanism to generate and process events.
- It provides simple concept.
  - A **source** generates an event and sends it to one or more **listeners.**
  - In this, the listener simply waits until it receives an event.
  - Once received, the listener processes the event and then returns.
- Advantage: application logic that processes events is cleanly separate from the user interface logic that generates those events. A user interface element is able to "delegate" the processing of an event to a separate piece of code.
- In delegation event model, listeners must register with a source in order to receive an event notification. Because notifications are sent only to that listeners which wants to receive them.
- In old version of java (1.0),event was circulated up the control hierarchy until it was handled by a component. This required components to receive event that they did not process, and it wasted valuable time. The delegation event model eliminates this overhead



- **Events**
  - When the user interacts with a GUI application, an event is generated.
  - Events are represented as Objects in Java. Changing the state of an object is known as an **event.**
  - Example: user events are clicking a button, selecting an item or closing a window. Events may also occur that are not directly caused by interaction with a user interface. An event may be generated when a timer expires, a counter exceeds a value, a software or hardware fails.
  - The java.awt.event package provides many event classes and Listener interfaces for event handling.

- **Event sources**
  - A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Source may generate more than one type of event.
  - A source must register listeners for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

    public void **addTypeListener**(TypeListener el)

    Here, Type is the name of the event and el is a reference to the event listener.
- **Event Listeners**
  - A listener is an object that is notified when an event occurs. It has two major requirements

  1) It must have been registered with one or more sources to receive notifications about specific types of events.
  2) It must implement method to receive and process these notifications.
  - The methods that receives and process events are defined in a set of interfaces found in java.awt.event.
- **Event classes**
  - Event classes represent the events. At root of Java event class hierarchy is **EventObject**, which is in java.util package.

    EventObject (Object src) //src is the object that generates event.
  - **EventObject** Contains 2 methods:

    1) Object **getSource** ( ): returns the source of the event.
    2) String **toString** ( ): returns the string equivalent of the event.
  - **AWTEvent** class defined in **java.awt** package is a subclass of **EventObject**. It is superclass (directly or indirectly) of all AWT-based events handled by delegation model.
  - Its **getID** ( ) method can be used to determine the type of event.

Main Event classes:

No.	Class	Description
1	**AWTEvent**	It is the root event class for all AWT events. This class and its subclasses supersede the original java.awt.Event class.
2	**ActionEvent**	The ActionEvent is generated when button is clicked or the item of a list is double clicked.
3	**InputEvent**	The InputEvent class is root event class for all component-level input events.
4	**KeyEvent**	On entering the character the Key event is generated

5	**MouseEvent**	This event indicates a mouse action occurred in a component.
6	**TextEvent**	The object of this class represents the text events
7	**WindowEvent**	The object of this class represents the change in state of a window.
8	**AdjustmentEvent**	Object represents When scroll bar is manipulated
10	**ComponentEvent**	The object of this class represents when component is hidden,moved,resized or becomes visible
11	**ContainerEvent**	The object of this class represents when component is added or removed from container
12	**MouseMotionEvent**	When mouse motion occurs ,mouse moved or dragged
13	**PaintEvent**	The object of this class represents when paint is invoked

1. ActionEvent class

- **ActionEvent is** generated when button is pressed, a list item is double clicked, menu item is selected or when press Enter key in a text box.
- This class is defined in **java.awt.event** package.
- **ActionEvent** class defines four integer constant that can be used to identify any modifiers associated with an action event.
  static int ALT_MASK -- The alt modifier.
  static int CTRL_MASK -- The control modifier.
  static int META_MASK -- The meta modifier.
  static int SHIFT_MASK -- The shift modifier.
  static int ACTION_PERFORMED -- This event id indicates that a meaningful action occurred

  Class declaration

  public class **ActionEvent** extends **AWTEvent**

  Class constructors

  1. **ActionEvent** (Object src, int id, String cmd): Constructs an ActionEvent object.
  2. **ActionEvent** (Object source, int id, String cmd, int modifiers): Constructs an ActionEvent object with modifier keys.
  3. **ActionEvent** (Object source, int id, String cmd,long when, int modifiers): Constructs an ActionEvent object with the specified modifier keys and timestamp.

  Class methods :

1. String **getActionCommand** ( ): Returns the command string associated with this action. For example, when a button is pressed, an action event is generated that has a command name equal to the label on that button.
2. int **getModifiers** ( ) :Returns a value that indicates which modifier keys (ALT,CTRK,META and/or SHIFT) were pressed when event was generated.
3. long **getWhen**( ) :Returns the time at which the event occurred
4. String **paramString**( ):Returns a parameter string identifying this action event.

2. MouseEvent class

- **KeyEvent** and **MouseEvent** are subclasses of abstract InputEvent class. Both these events are generated by objects of type Component class and its subclasses.
- **Class declaration**

  public class **MouseEvent** extends **InputEvent**
- **Fields**

  Following are the fields for java.awt.event.MouseEvent class:
  static int BUTTON1 --Indicates mouse button #1; used by getButton()
  static int BUTTON2 --Indicates mouse button #2; used by getButton()
  static int BUTTON3 --Indicates mouse button #3; used by getButton()
  static int NOBUTTON --Indicates no mouse buttons; used by getButton()
- **There are eight types of mouse events:**
  1. static int MOUSE_CLICKED --The "mouse clicked" event
  2. static int MOUSE_DRAGGED --The "mouse dragged" event
  3. static int MOUSE_ENTERED --The "mouse entered" event
  4. static int MOUSE_EXITED --The "mouse exited" event
  5. static int MOUSE_MOVED --The "mouse moved" event
  6. static int MOUSE_PRESSED -- The "mouse pressed" event
  7. static int MOUSE_RELEASED --The "mouse released" event
  8. static int MOUSE_WHEEL --The "mouse wheel" event

static int VK_WINDOWS --Constant for the Microsoft Windows "Windows" key.

- **Class constructor**

1. MouseEvent (Component src, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger):

Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, and click count.

- **MouseEvent class methods**
  1. int **getButton** ( ): Returns the value that represents the button that caused the event.
  2. int **getClickCount**( ) :Returns the number of mouse clicks for this event.

3. Point **getPoint**() :Returns the x,y position of the event relative to the source component.
4. int **getX**() :Returns the horizontal x position of the event relative to the source component.
5. int **getXOnScreen**() Returns the absolute horizontal x position of the event.
6. int **getY**() returns the vertical y position of the event relative to the source component.
7. int **getYOnScreen**() :Returns the absolute vertical y position of the event.
8. void **translatePoint**(int x, int y) Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets.
9. Point **getLocationOnScreen** (): Returns the absolute x, y position of the event.

3. WindowEvent class:

- **WindowEvent** are generated for the Window class and its subclasses. The object of this class represents the change in state of a window. These events are generated if an operation is performed on a window.
- **Class declaration**

  public class **WindowEvent** extends **ComponentEvent**
- **There are ten types of window events:**
  1. static int WINDOW_ACTIVATED --The window-activated event type.
  2. static int WINDOW_CLOSED -- The window closed event.
  3. static int WINDOW_CLOSING -- The "window is closing" event.
  4. static int WINDOW_DEACTIVATED -- The window-deactivated event type.
  5. static int WINDOW_DEICONIFIED -- The window deiconified event type.
  6. static int WINDOW_GAINED_FOCUS -- The window-gained-focus event type.
  7. static int WINDOW_ICONIFIED -- The window iconified event.
  8. static int WINDOW_LOST_FOCUS -- The window-lost-focus event type.
  9. static int WINDOW_OPENED -- The window opened event.
  10. static int WINDOW_STATE_CHANGED -- The window-state-changed event type.

- **Class constructors**
  1. **WindowEvent**(Window source, int id) : Constructs a WindowEvent object.
  2. **WindowEvent**(Window source, int id, int oldState, int newState) :Constructs a WindowEvent object with the specified previous and new window states.
  3. **WindowEvent**(Window source, int id, Window opposite) :Constructs a WindowEvent object with the specified opposite Window.
  4. **WindowEvent**(Window source, int id, Window opposite, int oldState, int newState) :Constructs a WindowEvent object.

- **Class methods**
  1. int **getNewState**() : events returns the new state of the window.
  2. int **getOldState**() : events returns the previous state of the window.
  3. Window **getOppositeWindow**() :Returns the other Window involved in this focus or activation change.
  4. Window **getWindow**() :Returns the originator of the event.
  5. String **paramString**() :Returns a parameter string identifying this event.

4. KeyEvent is subclasses of abstract **InputEvent** class. Both these events are generated by objects of type Component class and its subclasses. The KeyEvent is generated when the user presses or releases a key on the keyboard.

➢ Event Listeners

- The delegation event model has two parts: **source** and **listener**
- **Listeners** are created by implementing one or more of the interfaces defined by java.awt.Event.
- Every method of an event listener method has a single argument as an object which is subclass of EventObject class.

Most commonly used Event Listener interfaces

No.	Class	Description
1	**ActionListener**	This interface is used for receiving the action events.
2	**ComponentListener**	This interface is used for receiving the component events.
3	**ItemListener**	This interface is used for receiving the item events
4	**KeyListener**	This interface is used for receiving the key events.
5	**MouseListener**	This interface is used for receiving the mouse events
6	**TextListener**	This interface is used for receiving the text events.
7	**WindowListener**	This interface is used for receiving the window events.
8	**AdjustmentListener**	This interface is used for receiving the adjustment events.
10	**ContainerListener**	This interface is used for receiving the container events
11	**MouseMotionListener**	This interface is used for receiving the mouse motion events.
12	**FocusListener**	This interface is used for receiving the focus events.
13	**WindowFocusListener**	It is used for receiving event when a window gains or losses input focus

1. ActionListener interface

- The class which processes the **ActionEvent** should implement this interface.
- The object of that class must be registered with a component. The object can be registered using the **addActionListener** ( ) method.
- When the action event occurs, that object's **actionPerformed** ( ) method is invoked.
- **Interface declaration**

  public interface **ActionListener** extends **EventListener**
- **Interface methods:**

  void **actionPerformed** (ActionEvent e) // Invoked when an action event occurs.

2. MouseListener interface:

- The class which processes the **MouseEvent** should implement this interface.
- The object of that class must be registered with a component. The object can be registered using the **addMouseListener** () method.
- **Interface declaration**

  public interface **MouseListener** extends **EventListener**
- **Methods:**

1. void **mouseClicked** (MouseEvent e) :
2. void **mouseEntered**(MouseEvent e) :
3. void **mouseExited**(MouseEvent e):
4. void **mousePressed**(MouseEvent e) :
5. void **mouseReleased**(MouseEvent e):

3. Window listener

- The class which processes the **WindowEvent** should implement this interface.
- The objects of that class must be registered with a component. The object can be registered using the **addWindowListener** ( ) method.
- **Interface declaration**

  public interface **WindowListener** extends **EventListener**
- **Interface methods**

1. void **windowActivated** (WindowEvent e) :
2. void **windowClosed**(WindowEvent e) :
3. void **windowClosing**(WindowEvent e) :.
4. void **windowDeactivated**(WindowEvent e) :
5. void **windowDeiconified**(WindowEvent e) :
6. void **windowIconified**(WindowEvent e) :
7. void **windowOpened**(WindowEvent e) :

4. KeyListener interface:

- The class which processes the **KeyEvent** should implement this interface.
- The object of that class must be registered with a component. The object can be registered using the **addKeyListener** ( ) method.
- **Interface declaration**

  public interface **KeyListener** extends **EventListener**
- **Interface methods**
  1. void **keyPressed**(KeyEvent e) : Invoked when a key has been pressed.
  2. void **keyReleased**(KeyEvent e) :Invoked when a key has been released.
  3. void **keyTyped**(KeyEvent e) :Invoked when a key has been typed.

Event types and corresponding Event Source & EventListener

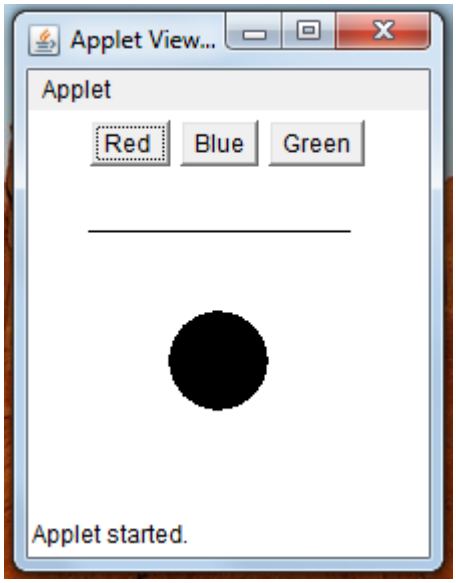
Event Type	**Event Source**	**Event Listener interface**
ActionEvent	Button, List, MenuItem, TextField	ActionListener
AdjustmentEvent	Scrollbar	AdjustmentListener
ItemEvent	Choice, Checkbox, CheckboxMenuItem, List	ItemListener
TextEvent	TextArea, TextField	TextListener
ComponentEvent	Component	ComponentListener
ContainerEvent	Container	ContainerListener
FocusEvent	Component	FocusListener
KeyEvent	Component	KeyListener
MouseEvent	Component	MouseListener, MouseMotionListener
WindowEvent	Window	WindowListener

Example:

Write a program to create applet with **one circle ,one label and three Button (Red,Blue,Green).**when you click on the button there should be change in color accordingly (circle shold be filled with hrespective color) and a label should display the selected color.

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

/*
<applet code=btn_click.class width=200 height=200>
</applet>
*/


public class btn_click extends Applet implements ActionListener
{
Label l;

        Button b1=new Button("Red ");
        Button b2=new Button("Blue");
        Button b3=new Button("Green");
        public void init()
        {
        b1.addActionListener(this);
                b2.addActionListener(this);
        b3.addActionListener(this);

        add(b1);
        add(b2);
        add(b3);

        l =new Label("                          ");
        add(l);
        }

        public void paint(Graphics g)
        {
        g.fillOval(70,100,50,50);
        g.drawLine(30,60,160,60);
```

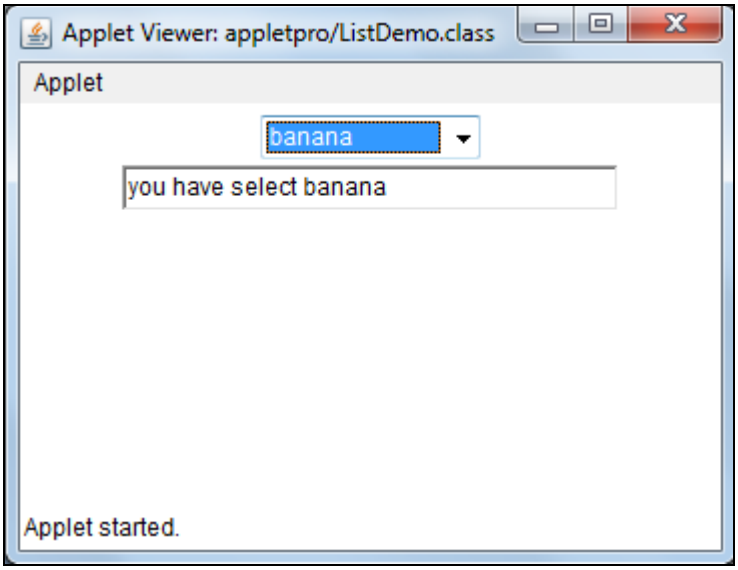
```
        }



        public void actionPerformed(ActionEvent e)
        {
        String s=(e.getActionCommand());
        if(s.equals("Red "))
        {
        setForeground(Color.red);
        }

        if(s.equals("Green"))
        {
        setForeground(Color.green);
        }
        if(s.equals("Blue"))
        {
        setForeground(Color.blue);
        }
        l.setText("This is "+e.getActionCommand()+" colour");
        }

}
```



Write a applet program which provides choice list and when user select any item it should be displayed in text box.

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

/* <applet code= ChoiceDemo.class width=300 height=500>
</applet>
*/

public class ChoiceDemo extends Applet implements ItemListener

{
TextField t1;
Choice c1;
Canvas can;
public void init()
{
c1=new Choice();
c1.addItem("select any Fruit");
c1.addItem("Apple");
c1.addItem("Orange");
c1.addItem("banana");

c1.addItemListener(this);

add(c1);

t1=new TextField(32);
add(t1);
t1.setText("No fruit selected");

}

public void itemStateChanged(ItemEvent e)
{
String s=c1.getSelectedItem();
if(s.equals("Apple"))
        t1.setText("you have select "+ s );
else if(s.equals("Orange"))
        t1.setText("you have select "+ s );
else
```

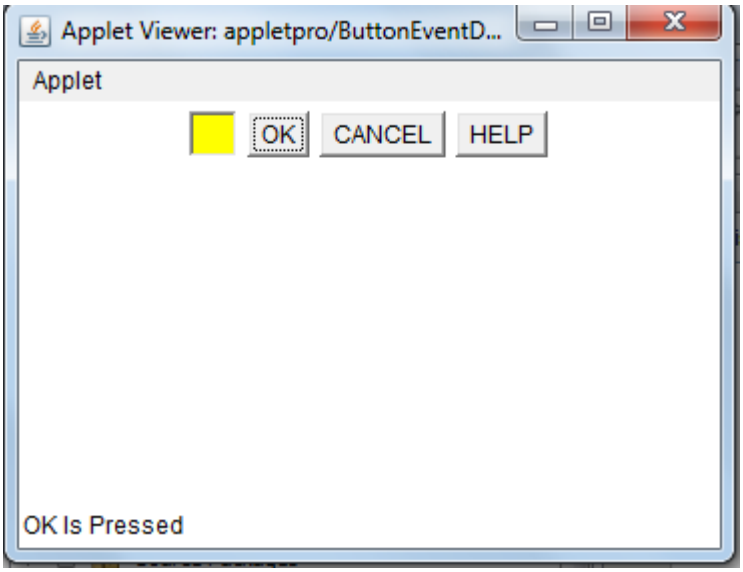
```
        t1.setText("you have select "+ s );

}

public boolean action(Event e,Object a)
{
repaint();
return true;
}
}
```



Write an applet that contains three buttons OK, CANCEL, AND HELP and a text field. If OK is pressed, then show on the status bar – "OK is pressed" and the text field turn yellow. When CANCEL is pressed, show on the status bar – "CANCEL is pressed" and the text field turn red. When HELP is pressed, show on the status bar – "HELP is pressed" and the text field turn green.

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

/*
<applet code="ButtonEventDemo" width=250 height=250>
</applet>
*/

public class ButtonEventDemo extends Applet
{
        TextField tf = new TextField();
```

```
        Button b1 = new Button("OK");
        Button b2 = new Button("CANCEL");
        Button b3 = new Button("HELP");
        public void init()
        {
                add(tf);
                add(b1);
                add(b2);
                add(b3);
        }
        public boolean action(Event e, Object o)
        {
                if(e.target.equals(b1))
                {
                        showStatus("OK Is Pressed");
                        tf.setBackground(Color.YELLOW);
                }
                else if(e.target.equals(b2))
                {
                        showStatus("CANCEL Is Pressed");
                        tf.setBackground(Color.RED);
                }
                else if(e.target.equals(b3))
                {
                        showStatus("HELP Is Pressed");
                        tf.setBackground(Color.GREEN);
                }
                return true;
        }
}
```



Write an applet which contains different colors in list.when user any color canvas color should be changed.canvas size should be 60X60 pixels.

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

/*
<applet code="ChoiceListEventDemo" width=250 height=250>
</applet>
*/

public class ChoiceListEventDemo extends Applet implements ItemListener
{	String s1;
	int i;
	Choice ch = new Choice();
	Canvas cn = new Canvas();
	public void init()
	{

		cn.setBackground(Color.RED);
		cn.setSize(60,60);
		add(cn);

		ch.addItem("Red");
		ch.addItem("Blue");
		ch.addItem("Green");
		ch.addItem("Pink");
		ch.addItem("Yellow");
		ch.addItem("black");
		ch.addItemListener(this);
		add(ch);
}
	public void itemStateChanged(ItemEvent e)
	{

		if(ch.getSelectedItem().equals(0))
		{
			cn.setBackground(Color.RED);
			showStatus("Red Is Selected");
		}
		else if(ch.getSelectedItem().equals("Blue"))
		{
```

```
                cn.setBackground(Color.BLUE);
                showStatus("Blue Is Selected");
            }
            else if(ch.getSelectedItem().equals("Green"))
            {
                cn.setBackground(Color.GREEN);
                showStatus("Green Is Selected");
            }
            else if(ch.getSelectedItem().equals("Pink"))
            {
                cn.setBackground(Color.PINK);
                showStatus("Pink Is Selected");
            }
            else if(ch.getSelectedItem().equals("Yellow"))
            {
                cn.setBackground(Color.YELLOW);
                showStatus("Yellow Is Selected");
            }
            cn.repaint();
        }

}
```

Quesions:

1. Explain following class: Frame, Panel, Canvas, and Window
2. Explain following AWT controls:
    a. Button
    b. TextField
    c. Checkbox
    d. Label
    e. List
    f. Choice
    g. Menu
    h. radio button
3. What is Layout Manager? List out layout manager. Explain any one with example.
4. Difference between Swing and AWT
5. Explain any three swing controls.
6. What is event? List out all Event classes.
7. What is event listener? List out all event listeners.