AWT (Abstract Window Toolkit)

How the O/S manages to provide to develop GUI programs:-

1. Every application came with main window.
2. Toolbar icon & menu bars will be appearing with every window.
3. Contains different types of GUI components.
4. Java people calls main window as a frame.

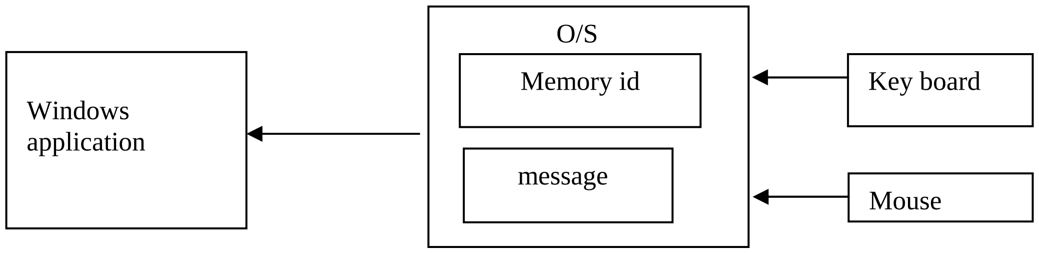
spg++ (comes with visual C++) is used how your O/S works at the back to develop GUI applications. This gives all the details of windows that contained other windows or managed by O/S at the back.

* The frame in which it is displayed is different that of your textbox / button.
* The operation/actions that can be performed by the frames is different that of textbox/button.

How the Input strokes are taken by the system and how this is managed by the O/S:-

Message = Event in java language.
Message is nothing but some information given to the windows by an O/S by selecting the appropriate window. O/S is responsible to take the inputs from the input devices/ external events.

Any O/S which supports windows acts like this:-



Windows application consists of different types of windows. The major differences between these windows are:
1. The way in which they get displayed.
2. The way in which they react to the events.

* Message is nothing but the structure in 'C'.

// UP TO 74 .....

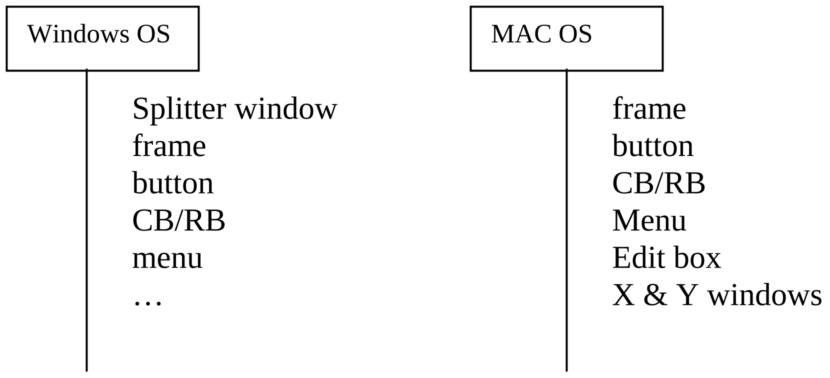
Message is stored according to the message id, which helps the pointer to the particular memory area to store appropriate message according to the message id in the constructor.

```
                Struct msg{     // structure in consturctor
                                                int msgid;
                                                OA*OA;                  }
```

When the mouse is moving, the O/S identifies that particular integer value that is assigned for a particular event and displays appropriate message for that id.

Message ids (or) event ids that are used by different windows applications are different, but the way in which they are working is the same.

Different events have different integer values to identify that particular event. According to this, the O/S knows that particular event has occurred through the O/S and will pass the appropriate message to the windows application. (What information is to get from that particular message will be managed by the O/S by taking the message id)



To develop Java platform independent, they write the code to work windows O/S that are not supported X and Y windows of MAC O/S.

They have taken all the common types of components/controls across multiple O/S and create a set of classes to develop platform independent. The set of classes are available in *java.awt* package.

java.awt consists set of classes which allows you to write GUI applications/programs. Java programmer should not know how the O/S manages the windows.

Later they have developed *javax.swing*/ *JFC swing*. These are developed because *awt* has limited number of controls/components. Java people have taken least common denominator to develop *awt* classes because they have very short time duration to release java as soon as in the market.

Microsoft has developed their own set of classes as AFC (Application Foundation Classes). These are more flexible than AFC's.

JavaSoft want to team up with the people of IBM & Netscape and they have started a project (before developing the project, one code name should be given to that project and later this can be changed according to their convenience. This is the standard that is used across the glove), with the name as swing. These fellows copied all the MFC from Microsoft and developed the set of classes as JFC, and now it is called as JFC swing.

JavaSoft people developed this to identify the java programmers, that they are specific from the other classes like MFC from Microsoft.

*Firstly windows applications/GUI applications are developed by MAC O/S which has provided more facilities than windows O/S. 15 years back they have developed the windows applications.

While designing the *awt* classes, JavaSoft has identified that all the GUI classes can share class *java.awt.component*. This is the super-class for all your GUI components.

There are two types of GUI components:

In *awt* package, we have another important class called container class. Examples of containers are frames, Dialog Boxes and panels.   Frame is a sub-class of a container class.

Container components/controls that allow us to place/add some other controls /components on it.

**Window* class is a super-class of a *Frame*-class, and *Dialog* class.

Window -- No border, no title bar (min, max, fclose buttons)
Frame -- Has border and title bar.

Dialog class: Initiate to the dialogue of the user. (i.e. it takes some input from the user and proceeds for the next)

Object window takes another window as a parameter.

```
          Window mywin1=new Window(mywin);
          mywin1.setSize(new Dimension(100,100));
          mywin1.setBackground(Color.orange);
          mywin1.show();
                    }
```

Observe the output here. (Output will be a window with orange background)

The *Window* class is mainly designed to act as a super-class for other classes. If you directly use a *Window* object, it will display a rectangular area without Title bar and border.

*   In *awt* package, we have a set of classes which provides the implementation of interfaces *LayoutManager* and *LayoutManager2*. These classes can be used with the containers and these are responsible for laying out the components. (Taking the decision of where to and how to display the components, and what is the size (width & height) of the components).

Any Container has to take the help of *LayoutManager*s in-order to layout the components.

* *setSize()* : This method layouts the size of the container according the user requirements, given as per the parameters.
* *pack()* : This method is going to take the help of the LayoutManagers and provides the appropriate size of the container and sets the size used to display the exact size of the container with the controls.
// UPTO 75

```
   f.setLayout(new FlowLayout());
  //displays the components.
```

In most of the cases we will prefer to write the code as:-

```
import java.awt.*;

public class UseButton{
public static void main(String args[]){
            MyFrame1     mywin1 = new MyFrame1("my owner frame");

                         mywin1.setSize (new Dimension (100,400));
                mywin1.setBackground (Color.orange);
                mywin1.show();
                mywin1.pack();
                                        }
                                }

class MyFrame1 extends Frame {
                     Button b1,b2;
                    Label l1;
                   public MyFrame1(String  s){
                            super(s);

                           b1=new Button("Button1");
                            b2= new Button("Button2");
                                       setLayout(new FlowLayout());
                            add(b1);
                                                 add(b2);
                                                 add(l1=new Label("This is a Label"));
                            pack();  //setSize should not be used with pack();
                                          }
                              }
```

/* Instead of writing in a Main-class, we are writing here as a separate sub-class.*/

* Create a frame object and add 3buttons and execute the *setBounds()* on these 3buttons.

Set the *LayoutManager* to null ones and in another experiment set the *LayoutManager* to *FlowLayout*?

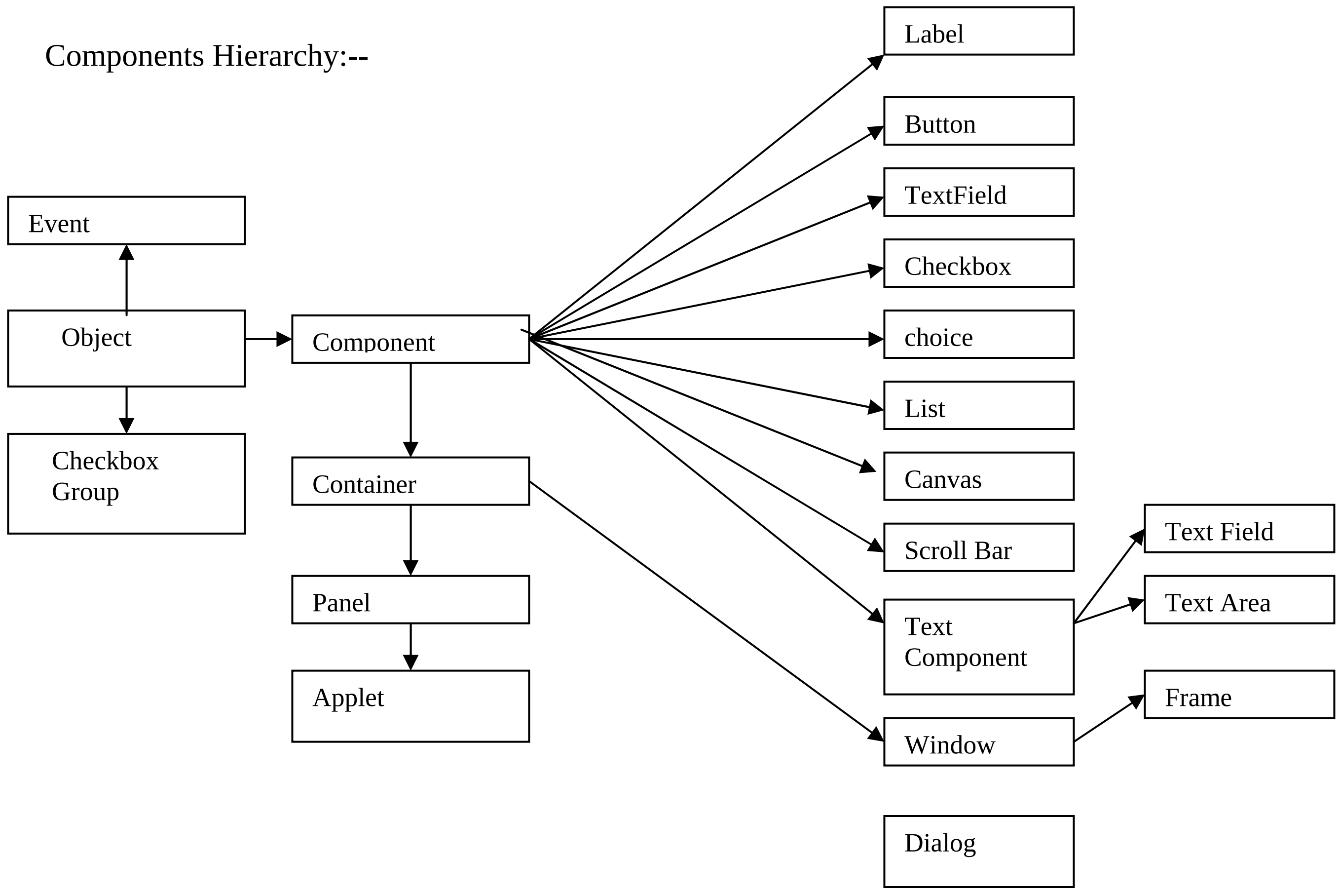
```
setBounds(Size ,Location);  //=  setSize(),setLocation;
setLayout(null);
b1.setBounds(100,100,50,50);                    diag:
b2.setBounds(200,200,100,100);
```

* *setBounds* will not effect in the case of *FlowLayoutManager*. Whenever a LayoutManager is used, the *LayoutManager* decides the position and size of the components inside a container. If you want to programmatically set the size and location, we can use a null *LayoutManager*.

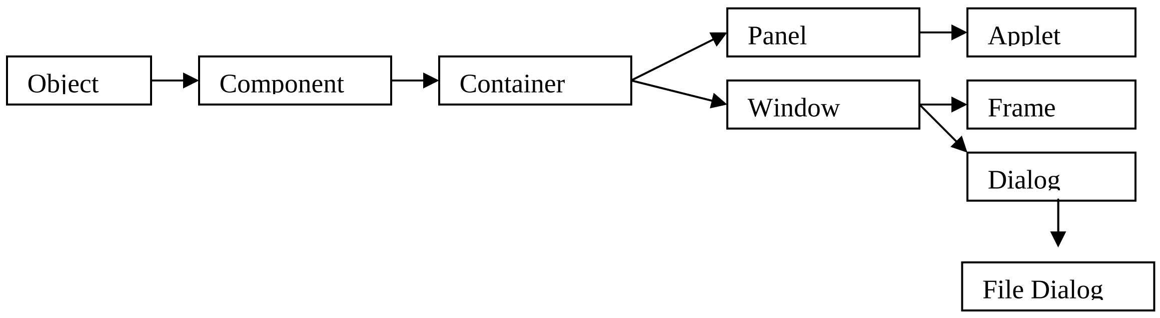
But it is not recommended to do so.

Components: Are java's building blocks for creating Graphical User Interface (GUI'S). Components are placed onto a user interface by adding them to a container.

Components Hierarchy:--

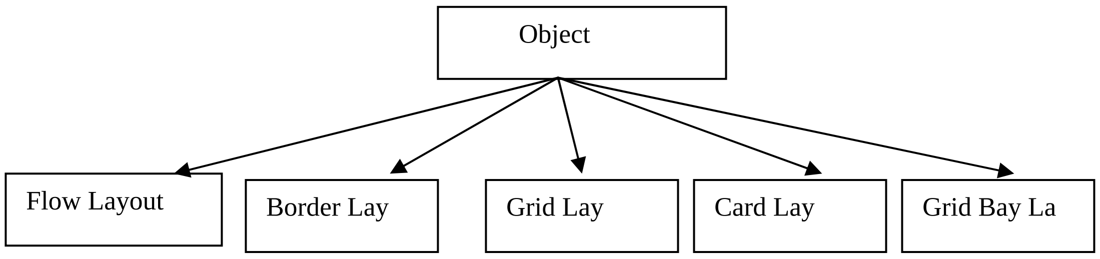


Container Hierarchy:-
Container and components are abstract classes. Container holds the components.
Eg:- window, frame, applet, panel, dialog etc......



* Frame and Applet contains border and can act as top-level windows.
* Panel does not contain any border, for his reason panels cannot act independently and always works inside a window or applet. Panels cannot be a top level window.

LayoutManagers Hierarchy:



There are two options in lying the components in a container.

1) By specifying the X and Y co-ordinates on location like north or south etc. If X & Y used, then the GUI becomes system dependent (because X & Y depends on screen resolution). Suppose a button is places using x,y at centre inside a container by calculating its width and height. If frame is resized, the button looses its center position and remains in the original position only and does not act to be at center again.

2) To overcome this problem, java people introduced *LayoutManager*s system. In this system component is not added by X,Y but the location is specified by using different *LayoutManagers*.

When we programmatically position the components, it is the responsibility of the developer to reposition the components location.

JavaSoft provided the implementation of *LayoutManagers* in different classes like *FlowLayoutManager*, *CardLayoutManagers* and so on. If these classes don't fit our requirements, we can create our own classes which provides *LayoutManager* interface.

/ Demonstrates how to show window*

```
import java.awt.*;
import java.awt.event.*;
public class PaintTest
{
        p.s.v.m(String args[]) throws Exception
        {
                Myframe mywin=new Myframe("My new Frame");
                mywin.setSize(new Dimentsion(200,200));
                System.out.println("Number of threads="+Thread.activeCount());
                mywin.show();
                Thread.currentThread().sleep(10000);
                for(Myframe.i=0;Myframe.i<100;Myframe.i++)
                {
                        Thread.currentThread().sleep(1);
                        // System.out.println("..........before Repaint..........");
                        mywin.repaint();
            }
        }
}

class Myframe extends Frame
{
        public static nt i=0;
        Event e;
        public Myframe(String s)
        {
                super(s);     // To override the frame name
        }
        public void paint(Graphics g)
        {
                System.out.println("In paint event");
                g.drawString(" In paint event",100,100);
        }
        public void update(Graphics g)
        {
                System.out.println(i+ "*** in update event");
                super.update(g);   // calls paint event
                //paint(g);       }         }
```

Note: Observe the output carefully.

In a component class, there is a paint method .This is called whenever painting is required on the component. When you want to display the component first time, then the *paint()* method is called [*frame1.show()*].If you overlap the component by another component or when you drag the frame or when you resize the frame, then *paint()* method is repainted.

Whenever a *paint()* method is called the component, it is responsible by the graphics class where to paint the component.
(graphics-->Using which the programmer performs graphics)

Every GUI is associated with an object of type '*Graphics*'. This object provides the useful methods to decide about where the graphics to be performed and it provides several methods, which helps us in performing the drawing operation on the component.
In almost all the components some space will be reserved for drawing the borders.
Insets: Amount of space reserved to draw the *borders(top, left, bottom, and right insets.)*.We can find out this information by *getInsets()* method.

Constructor Summary:-
Insets(int top, int left, int bottom, int right)
Creates and initializes a new Insets object with the specified top, left, bottom, and right insets.

When we performing the painting operation using *Graphics* object, this will takes care about the boundaries of the component .If we try to paint outside the boundaries, this *paint()* method should not display anything.

```
        g.setColor(Color.red);
        g.drawString("In paint Event",200,200);
```

The *setColor()*method can be used to set the current color of the Graphics object. Whenever a Graphics operation is performed, the graphics object uses current color. Similarly font can set by using *setFont()* method.

* You should not directly call the *paint()* method whenever the painting has to be performed, you just call repaint() method, it will take care about calling the paint() method.

```
        mywin.repaint();
```

* The Component class consists of three methods, *repaint(),update()* and *print()* method.
* Whenever we override the *update()* method, it is the responsibility of the programmer to call the *print()* method.
* The default implementation of the *update()*method (update method in a component class) is responsible for clearing the background.

update() method will be called only when there is a pending requests of repaint() method.
*   In some cases, we may call the *repaint()* operation repeatedly which may not call the update()method every time. It will calls the update method only ones to perform the pending requests as shown in the example PrintTest.java

```
class labelDemo extends frame
{
```

```
Label lab1,lab2,lab3;
Label Demo()
{
setLayout(new firstLayout());
Lab1=new Label1("Label");
Lab2=new Label("Label2");
Lab3=new Label();
Lab3.setText("label3");
Lab1.setAlignment(Label.LEFT);
lab3.setAlignment(Label.RIGHT);
add(LAb1);
add(LAb2);
add(Lab3);
setTitle("Labels Dosen't havw actions");
setSize(450,200);
setVisible(true);
}
public void paint(Graphics g)
{
g.drawString("Lab1.text:"lab1.getText(),50,100);
g.drawString("lab1 Alignment :"Lab1.getAlignment,50,120);

}
}
```

usechoice.java

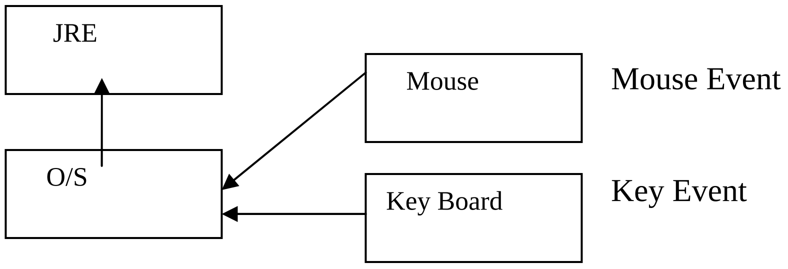
```
import java.awt.*;
import java.awt.event.*;

public class UseChoice{
        p.s.v.main(String args[]){
                MyFrame mywin=new MyFrame("Emp Information");
                mywin.setLayout(new FlowLayout());
                mywin.pack();
                mywin.show();
        }   }
class MyFrame extends Frame{
        Choice mychoice;
        Panel p1,p2;
        Checkbox regularemp,female,above30,abovenormalheight;
        public MyFrame(String title){
                super(title);
                p1=new Panel();
                p2=new Panel();
                mychoice=new Choice();
                p1.add(mychoice);
                mychoice.addItem("Bsc : COMPUTERS");
                .
                .
                .
                mychoice.addItem("Bsc : STATS");
                female=new Checkbox("Female",true);
                above30=new Checkbox("Above30");
                //we need not explicitly set value to false
                abovenormalheight=new Checkbox("Above Normal Height");
                p2.add(female);
                p2.add(above30);
                p2.add(abovenormalheight);
                add(p1);
                add(p2);
        }
}
```

Note:- Observe the Output.

Your Operating System sees JVM as a normal program. When a user clicks the mouse or keyboard keys, a message is passed to the program through Operating System. This operation is known as "User Events".

When a mouse is clicked or when a key is pressed, then the Operating System is responsible to take those actions and sends the appropriate message to the Java run-time environment. In Java language messages are known as Events.



The Operating System is responsible to send an appropriate event to the JRE when a user clicks the mouse or when he presses a key and so on.

There are so many classes which represent the types of events the naming convention is also very simple to remember and to use these classes.

Each and every event represents an object. Documentation clearly explains that what are the events that are occurred on that particular component and when they occur.
In JFC and AWT different types of events are supported and all the classes has a very simple naming pattern xxxEvent (where xxx represents the type of event).

For example, we have the classes with the names *MouseEvent*, which is used to represent both *MouseUp* and *MouseDown* Events .Another class *MouseMotionEvent* is used to represent the event *Mouse Motion*. keyEvent is used to represents the events KeyUp and KeyDown. Like this there are several types of event classes which are part of the AWT Package. Whenever we use a particular component in a program, we need to first know the events that are supported by a component and when these events will be triggered (or) get fired (or) occurred.

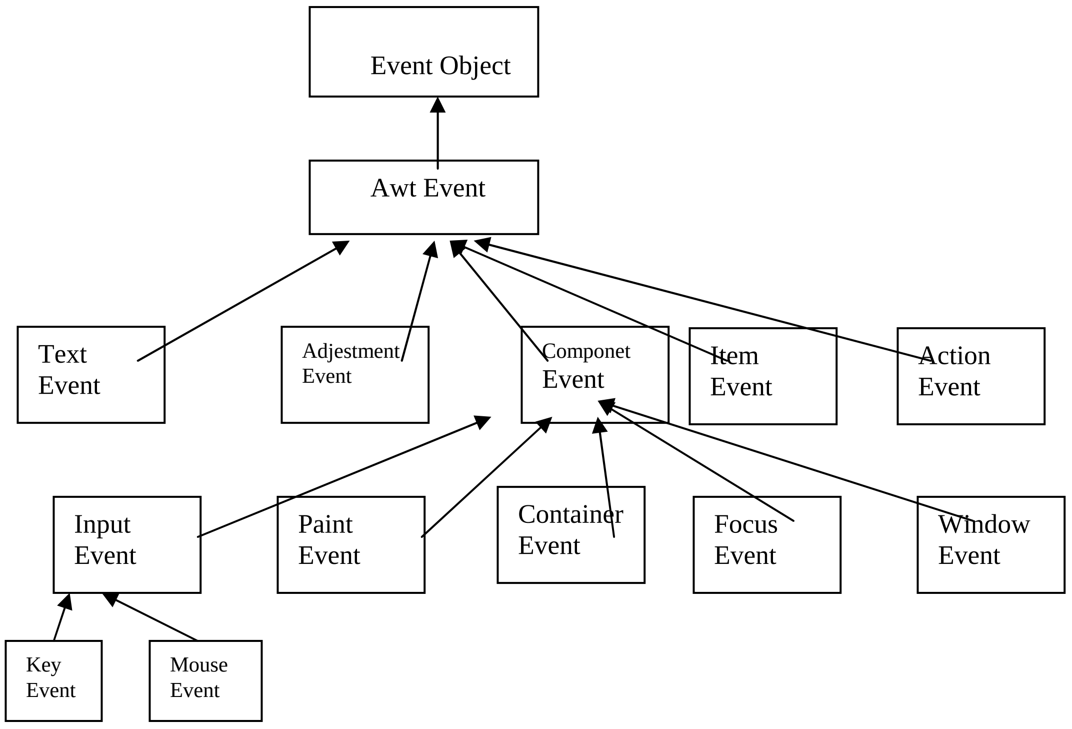
```
save =new MenuItem("Save");
saveas=new MenuItem("SaveAs");
e.add(save);
e.add(saveas);
help=new MenuItem("Help");
about=new MenuItem("About");
h.add(help);
h.add(about);   } }
```

Semantic Event: Ex:-Action Event

Has different meaning for different components. The documentation specifies when the event occurs.

Ex:-In a Text/TextArea event occurs when our Enter Key is pressed.
For a Button event occurs when the Button is clicked.
For an option/Radio Button ->clicked (True/False)
For Checkbox ->clicked or checked(0/1).

Events Hierarchy:-



At any point of time, only one component will have the focus. Operating System is responsible for rooting the events on that particular component/object. This will occur when the component gain or lost the focus.

The code which reacts to the action events is known as "Event Handlers". There are two types of event handlers in Java language.

1. Newer
2. Older

For every event type there is a corresponding xxxListener.xxx is an Interface corresponding to the event.

Ex:-
For ActionEvent -> ActionListener
For WindowEvent -> WindowListener
List of methods as part of the listener will be in the documentation.

Steps to create Events on a particular component:-

1) Implement the ActionListener Interface
2) Execute the method xxxListener by passing an appropriate Listener on the corresponding component/object.
3) Handle the events and impliment the ActionListener and execute this.
4) Display the message according to the event handlers of that particular component/object.

Note:-See the ActionListener Example

Whenever we want to write an event handler for a particular event, we need to provide the implementation of the appropriate Listener and then execute addxxxListener on the corresponding component/object.

toString() method generally used for debugging purpose.
toString() method used to override and to display the main properties of a particular component.

This can be used as part of xxxListener if we are providing the constructor.

Whenever an event occurs, the JRE is responsible for calling the listener that is added to the component.

In some cases, we will be adding the same Listener for multiple components. But we can identify the component on which the event has occurred by using the getSource() method.

In some cases, multiple listeners can be added to the same component. In this case a programmer should not rely on the sequence in which the listeners are intimated.

Create a class by providing the implementation of the interface WindowListener and add this WindowListener to a Frame.

When an AWT is used at the Operating System level for each and every java component there is one window will be managed by the Operating System (In Javas implimentaion)

AWT design is not a peer-component based.

AWT implementation by JavaSoft is a peer-component based (code written by using c+, Java). But the AWT implementation of Microsoft is not a peer-component based.

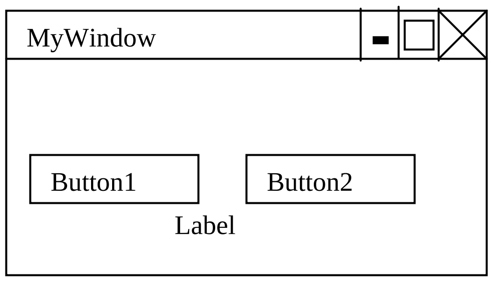
NOTE: Actually speaking AWT is not a peer based but the AWT implementers used this in their design according to their requirements and wish.

For Microsoft implementation of AWT, the windows are managed by the code written entirely in java language but not by the Operating System.

The Operating System comes with the code to manage all the components .But in Java (Jview) implementation, java people writes the wrapper (thin Layers) classes around the code that is managed by the Operating System i.e. for every component acts as a window at Operating System level.

NOTE: Using Ms-spy++, we can observe the difference between the different implications of AWTs.

If we observe the output of the figure using Spy++ by using MS implementation of AWT, it will shows 4 windows. But the same when we use Sunsoft's implementation of AWT, it will show only one window. Because here the code with an Operating System manages every component as a window at the back. But this is not done by the MS implementation of AWTs'.



JavaSoft people gone for this design, because:

They need not write too much of code.

Because all the components are placed on a (frame) single window, management of components is easier.

These people have very less time to release the Java in the market.

Difference between AWT and JFC:-

According to the books and JavaSoft people AWT components are heavy weight. (when a component consuming more recourses, then it is known as "Heavy Weight Component") why because when a component is created in Java language, it will create one peer component for every component at the Operating System level. So JavaSoft people have given this kind of statement.

JFC is light weight, because they don’t have a peer component at the Operating System level, because JFC (Java Foundation Classes) are entirely written in Java language.

Actually AWT components are light weight, and this will takes less memory than JFC. Try one example by using MS-Spy++, you know this.

JFC is a very flexible design. Using JFC components we can display the components look and feel. The design is so flexible that it allows us to add our own look and feel, i.e. we can add extensions to JFC classes to display different components according to our own choice. This is not possible in AWTs.

NOTE:-Experiment with MS-Spy++ by creating 100 buttons both in JFC and AWT.JFC will take much time to run, because the entire code is written in Java language and this code is loaded to the JVM and it is responsible to run the entire code. But AWTs are written in C++, Java languages. C Language runs much faster than JFC or Java. So, AWTs run faster than JFCs.

The way in which the classes are implemented and they are used is different in AWT and JFC.

To "Close the window" that is displayed during the runtime of a Java Program:-

At the end of your program you should write the following code:

```
Public static void main(String args[]){
        new classname();//creates an anonymous object to call the constructor
}
public void windowClosing(WindowEvent e){System.exit(0);}//Closes the Window
public void windowClosed(WindowEvent e){}
public void windowOpened(WindowEvent e){}
public void windowActivated(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowDeiconified(WindowEvent e){}   }
```

```
//Easiest way of choosing a window :using adapter class and inner classes

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

In the WindowListener Interface we override 'this' abstract method only.

WindowListener defines 7 methods in which one is overridden and remaining 6 abstract methods are simply overridden with empty braces as a rule.

```
C:\set PATH=C:\jdk1.4.2_03\bin;%PATH%;
```

While developing the GUI applications, the main point should remember is that taking less number of (amount) information from the inputs\user. If the input occurrences are more then set that particular object value as a default value.

Ex: - Radio Buttons, Check boxes and Choice boxes.

If you want to provide the user number of choices then go for Choice box (Combo box).

RadioButtons are not the separate components in java. If we want the RadioButtons then use CheckBoxGroup. A CheckBoxGroup groups a number of check boxes into a (group) single unit. If check boxes are in a group, we can select only one out of the group. Due to this nature, these are called as a "Radio Buttons".

Checkboxes:-Generally these are used to select between two choices. These are two state buttons that can be either "ON"(selected true) or "OFF"(Unselected false).When the user clicks the checkbox, the checkbox state changes and it generates an item event. This accomplished with a label indicating for what the checkbox is meant.

Choice:-is like a combobox where the options (choices) are popped out when the component is clicked. User can select only one item from the items displayed like radiobuttons. When an item is selected, it generates an item event.

List:-Unlike choice, list allows us to select more than one item at a time. That is list allows us to select multiple selections .List adds the scrollbar if the number of items exceeds the number of items for which list is set.

TextField :-after the Button, most sought is TextField. TextField and TextArea are the subclasses of TextComponent. TextField allows a single line of input from the user pressing Enter key in the TextField generates actionEvent.

TextArea:-Unlike TextField, TextArea permits to enter a number of lines(multi-line text)of text as input.TextArea adds ScrollBars by default. Textarea listens to text events.

ScrollBars:-Scrolls through a range of integer values. These are of two types: vertical and Horizontal. It has the minimum and maximum values and the initial value which will be set when displayed first time.

This Listens to AdjustmentListener Interface; the only method we have to override is public void adjustmentValueChanged(AdjustmentValue e)

Menus:-
Menu is not a component because it is not a subclass of Component class.Menu, MenuItem etc are derived from MenuComponent class which is again a subclass of object.

Menu Hierarchy:-

MenuBar holds the Menus. MenuBar is set to the Frame.MenuBar cannot be added to panels.
MenuItems are capable to generate both ActionListener(to create action) and ItemListener(to get the selection).

Steps to create Menus:-

1) Create a menubar(MenuBar mb= new MenuBar();)
2) Add the menubar to the frame(setMenuBar(mb);)
3) Create menus(Menu file=new Menu("File");)
4) Add items to each Menu(file.add(new MenuItem("OPEN");)
5) Add each menu to MenuBar(mb.add(file);)

A click on MenuItem generates an action event.Checkboxmenuitems are different from MenuItems by providing a checkbox next to MenuItem.Like checkboxes, checkboxMenuItems includes two methods: setState() and getState() to set and to get the state of checkbox menuitem.

Canvas: - is a component that has no default appearance or behavior. To create a Canvas we must subclass a Canvas class. This receives input events from the mouse and the keyboard. It is to the programmer to convert these input events into meaning full look and feel. Canvas objects are useful as display areas for images and custom graphics.

Panels:-is a subclass of a container. It allows you to add the components. We can set one component to one panel and more than one can also set to one panel and so on. Panel is used to effectively control the objects that are placed in the panel are treated as a single object. So, if you reduce the size of a window the objects that are placed on a panel will affect and they will appear half or in same cases some objects will not appear.

Panel is frequently used in the GUI design to group a set of components and control the layout. Very big advantage of a panel is components can be controlled very easily.

Modify usechoice example by creating a third panel add panel1 and panel2 to the 3rd panel, and add this 3rd panel to the frame.

for TextArea   ScrollbarsAlways
                ScrollbarsNone
                ScrollbarsWhenRequired
are the parameters that are passed according to your convenient that whether you need scrollbars or not.
Frame itself comes with a panel. If you want to add more panels, you can add according to your requirement.

Modify the useLIst example to display the text area with the ScrollbarsWhenRequired. And add a button to the same example and write an even handler on the button to exchange the text between textarea and textfield and observe the result.

Result:- Text that is entered in a TextField is exchanged to the TextArea. But the text from TextArea to TextField is not exchanged.

Cascaded Menus:- Adding the menu's to another menu to have multiple levels of menus.

appmenu.setHelpMenu(Help); --> this will places the help menu always at the last/end. This will be displayed according to the JRE implementation of different implementers. So this can displays the output differently in different platforms.

file.addSepeartor(); --> to add the sepearator between the menu items.

PopupMenu : - by right clicking the mouse we can get the popup menus in windows operating system. This will popup the user to quicly the selected items.

ShortcutKeys :- These are used to open the menu items quickly by pressing the combination of keys from a keyboard.

```
MenuShortcut ms1,ms2;
ms1=new MeneShortcut('o');   ---> ctrl+o
ms2=new MenuShortcut('e', true);   ---> ctrl+shift+e
newf.setShortcut(ms1);  // to add the shortcut keys.
save.setShortcut(ms2)  // to a menu items.
```

if we use true, it will assigns the shortcut key with 'shift' option.

For a convenient mechanism for the developers, JavaSoft created a set of classes with the name xxxAdapters. Where xxx represents name of that particular adapters.
Eg:- MouseAdapter(); used to implement the MouseListener. Here we see the balnk implementation (to get the popup menus this will be used).

Blank implementations of methods privided by MouseListener, whenever a programmer wants to handle one or two events of type mouse events. Then he can simply creates a sub-class of MouseAdapter() and selectively overridse the appropirate methods.

To make the Java on different platforms the display popupmenus, they used the isPopUpTrigger() method.

A popup menu on windows operating system has to displayed when the user clicks on the right button. But where as in some other platforms we need to display the popup when the user presses simultaneroulsy left and right buttons. And in the future operating systems we may need to do the same by clicking all the 3 buttons. In order to mask this platform dependent issue javasoft provides a simple method to detect whether the popup to be displayed or not using the isPopupTrigger() method.

ButtonDemo.java

```
import java.awt.*;
import java.awt.event.*;
public class ButtonDemo extends Frame implements ActionListener, WindowListener{
                Button b1,b2;
                ButtonDemo(){  //getting the benefirs of constructors for automatic
                                          initialization of different activities
                setLayout(new FlowLayout());
                b1=new Button("Green");
               b2=new Button("blue");
               b1.addActionListener(this); // to generate events
               b2.addActionListener(this); // listener is added
               add(b1);
               add(b2);
               setTitle("Button with actions");
               setSize(360,200);
              addWindowListener(this);
          }
          public void actionPerformed(ActionEvent e){
                    String str=e.getActionCommand(); // gets the label of the button clicked.
                    if(str.equals("Green");{
                    setBackground(Color.green);
                    b1.setForeground(Color.white); // this is equalent to setColor() of
graphics class
                    }
                    elseif(str.equals("Blue")){
```

```
		setBackground(Color.blue);
		b1.setForeground(Color.red);
		}
		if(b=b1){
			setBackground(Color.green);
			b1.setForeground(Color.while);
		}
		elseif(b=b2):
			setBackground(Color.blue
			b1.setForeground(Color.red
		}
	}
	public static void main(String args[]){
		new ButtonDemo(); //create an anonymous object to call the constructor.
	}
	}
```

CheckboxDemo.java

```
	import java.awt.*;
	import java.awt.event.*;
	public cl;ass CheckboxDemo extends Frame implements ItemListener,
ItemSelectable{
		Checkbox cb11,cb2,cb3;
		Label l1;
		CheckboxDemo(){
			setLayout(new FlowLayout());
			cb1=new Checkbox("dialog");
			cb2=new Checkbox("Bold",false);
			cb3=new Checkbox();
			cb3.setLabel("Italic");
			cb3.setState(flase);
			l1=new Label("AWT componenets are easy to learn"):
			cb1.addItemListener(this);
			cb2.addItemListener(this);
			cb3.addItemListener(this);
			add(cb1);
				add(cb2);
			add(cb3);
			add(l1);
			setTitle("checkbox demo");
			setSize(300,200);
			setVisible(true);
			addWindowListener(this);
		}
		public void itemStateChanged(itemevent e){
```

```
                int bold,italic;
                String str;
                if(cb1.getState()==true)
                        str="Dialog";
                else
                        str="monospeed";
                if(cb2.getState()==true)
                        str=Font.BOLD;
                else
                        str=Font.PLAIN;
                if(cb3.getState()==true)
                        italic=Font.ITALIC;
                else
                        italic=Font.PLAIN;
                l1.setFont(new Font(str+bold+italic,15);
        }
        public static void main(String args[]){
                new CheckboxDemo();
        }
}
```

try the same example with CheckboxGroup (3 radio buttons)

```
CheckboxGroup cbg=new CheckboxGroup();
cb1=new Checkbox("Red",cbg,true);
```

TextFieldTextArea.java

```
        import java.awt.*;
        import java.awt.event.*;
        public class TFTA extends Frame implements ActionListener, TextListener{
                TextField tf;
                TextArea ta;
                TFTA(){
                        SetLayout(new FlowLayout());
                        tf=new TextField("Rajest - How r u?", 50);
                        //columns width creates a textfield of 50
                        ta=new TextArea("Enter your Resume",10,20);
                        //creates a text area of 10 rows and 20 columns
                        tf.addActionListener(this);
                        ta.addActionListener(this);
                        add(tf);
                        add(ta);
                        setTitle("Text field and TextArea");
```

```
			setSize(450,350);
			setVisible(true);
			addWindowListener(new WindowAdapter()){
				public void WindowClosing(WindowEvent e){
					System.exit(0);
				}
			}

		}
		public void actionPerformed(ActionEvent e){
			repaint();
		}
		public void textValueChanged(TextEvent e){
			repaitn();
		}
		public void paint(Graphics g){
			g.drawString("tf.getText():"+tf.getText(),50,200);
			g.drawString("ta.getText():"+ta.getText(),50,220);
			g.drawString("tf.getColumns():"+tf.getColumns(),50,240);
			g.drawString("ta.getColumns():"+ta.getColumns(),50,260);
			g.drawString("tf.getRows():"+tf.getRows(),50,280);
			g.drawString("ta.getRows():"+ta.getRows(),50,300);
			g.drawString("tf.isEditable():"+tf.isEditable(),50,320);
			g.drawString("ta.isEditable():"+ta.isEditable(),50,200);
		}
		public static void main(String args[]){
			new TextFieldTextArea();
		}
	}
```

ListDemp.java

```
	import java.awt.*';
	import java.awt.event.*;
	public ListDemo extends Frame implements ActionListener, ItemListener{
		List tiffin;
		double price[]={6.00,10.00,9.50.10.50,8.00,6.00};
		// an array of prices for tiffin items
		String names[];
		int nos[];
		ListDemo(){
			setLayout(new FlowLayout());
			tiffin=new List(5,true);
			// no of visible rows is 5 and set to multiple selection code
			tiffin.add("idly"); // addItem() is deprecated in list
```

```
                tiffin.add("dosa");
                tiffin.add("wada");
                tiffin.add("puri");
                tiffin.add("bonda");
                tiffin.add("upma");

                tiffin.setBackground(color.cyan);
                setBackground(Color.yellow);
                tiffin.addActionListener(this);
                add(new Label("select your dish:");
                add(tiffin);
                setTitle("arpitha Hotel");
                setSize(500,450);
                setVisible(true);
                addWindowListener(new WindowAdapter()){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                        }
                }
                public void actionPerformed(ActionEvent e){
                        names=tiffin.getSelectedItems();
                        nos=tiffin.getSelectedIndexes();
                                repaint();
                }
                public void paint(Graphics g){
                int y=180;
                double sum=0.0;
                g.drawString("SL.NO      Itemname       Itemcode  Rate",50,150);
                        for (int i=0;i<names.length;i++,y+=25){
                        g.drawString((i+1)+"    "+names[i]+"    "+nos[i]+"
"+price[nos[i]],50,y);
                                sum=price[nos[i]];
                        }
                        g.setFont(new Font("dialog",Font.BOLD,15));
                        g.setColor(Color.magenta);
                        g.drawString("please pay Rs. "+sum,50,y+=35);
                        g.setColor(Color.red);
                        g.drawString("if dishes are delicous, please vising again"
+ 50,y+=35);
                }
                public static void main(String args[]){
                        new ListDemo();
                }
        }
```

note: list generates two types of events:

1. action events for double clicks --->ActionListener
2. ItemEvents for single click ----> ItemListener

In this example only ItemListener is used.

```
	public void itemStateChanged(ItemEvent e){
		Selection=tiffin.getSelectedItem();
		repaint();
	}
	public void paint(Graphics g){
		int y=210;
		g.drawString("Double clicked items ====" 50.180);
		for (int i=0;i<names.length;i++,y+=20){
			g.drawString(names[i],50,y);
			g.drawString("single clicked item name 
:"names.getSelectedItems(),50,y+=20);
			g.drawString("single clicked item index :" 
names.getSelectedIndex(),50,y+=20);
			g.drawString("list is set to multi-
selection:"+tiffin.isMultipleMode(), 50,y+=20);
			g.drawString("total no.of items:"+tiffin.getItemCount(), 
50,y+=20);
			g.drawString("no. of visible items:"+tiffin.getRows(),50,y+=20);
			g.drawString("7th item in the list "+tiffin.getItem(6),50,y+=20);
		}
	}
```

add this code to the previous example and observe the result.

CanvasDemo.java

```
	import java.awt.*;
	import java.awt.event.*;
	public class CanvasDemo extends Frame{
		Canvas c1,c2,c2;
		CanvasDemo(){
			setLayout(new FlowLayout());
			Label l1=new Label("Red Canvas");
			Label l1=new Label("green Canvas");
			Label l1=new Label("blue Canvas");

			c1=new Canvas();
			c1.setBackGround(Color.red);
```

```
                    c2=new Canvas();
                    c2.setBackGround(Color.green);

                    c3=new Canvas();
                    c3.setBackGround(Color.blue);

                    c1.setSize(100,100);
                    c2.setSize(100,100);
                    c3.setSize(100,100);

                    add(l1); add(c1);
                    add(l2); add(c2);
                    add(l3); add(c3);

                    setTitle("just a canvases");
                    setSize(250,400);
                    setVisible(true);

                    addWindowListener(new WindowAdapter(){
                        public void WindowClosing(WindowEvent e){
                            System.exit(0);
                        }
                    });
                public static void main(String args[]){
                    new CanvasDemo();
                }
            }
            }
```

ScrollbarsDemo.java

```
            import java.awt.*;
            import java.awt.event.*;
            public class ScrollbarsDemp extends Frame implements AdjustmentListener{
                Label lr,lb,lg; //red,green,blue labels
                Scrollbar redsb,greensb,bluesb;
                Canvas c;
                ScrollbarDemo(){
                    Panel p=new Panel(); // panel to contazin labels and scrollbars
                    p.setBackGrount(Color.cyan);
                    p.setLayout(new GridLayout(3,2,6,6));
                    p.add(lr=new Label("Red");
                    lr.setBackGround(Color.red);
                    p.add(redsb=new Scrollbar(Scrollbar.HORIZONTAL,0,0,0,255));
                    redsb.setUnitIncrement(5);
```

```
        redsb.setBlockIncrement(15);
        p.add(lg=new Label("green");
        lg.setBackground(Color.green);


        p.add(bluesb=new Scrollbar());
        bluesb.setOrientation(Scrollbar.HORIZONTAL);
        bluesb.setUnitIncrement(5);
        bluesb.setBlockIncrement(15);

        redsb.addAdjustmentListener(this);
        greensb.addAdjustmentListener(this);
        bluesb.addAdjustmentListener(this);

        add(p,SOUTH);  //adding the panel to the south of the frame
        c=new Canvas();
        c.setBackGround(Color.black);
        add(c,"center");
        setTitle("16 million shades for your choice");
        setSize(400,400);
        addWindowListener(new WindowAdpater(){
                public void windowClosing(WindowEvent we){
                        System.exit(0);
                }
        });
}
public Insets getInsets(){
        setBackground(Color.magenta);
        return new Insets(10,10,10,10); //equal space of 10 pixels each
}

public void adjustmentValueChanged(AdjustmentEvent e){
        int redqty=redsb.getValue();
        int greenqty=greensb.getValue();
        int blueqty=bluesb.getValue();

        lr.setText("Red"+redqty);
        lg.setText("Green"+greenqty);
        lr.setText("Blue"+blueqty);

        Color clr=bew Color(redqty, greenqty, blueqty);
        // color object is created with the current values of scrollbars
        c.setBackground(clr);
        c.repaint();
}
public static void main(String args[]){
```

```
                new ScollbarsDemo();
        }
}
```

Insets class creates space between the components and the border (not between the components) of the frame.
AdjustmentListener interface declares only one abstract method adjustmentValueChanged(AdjustmentEvent e).

Submenus and Shortcut keys:-

```
Menu submenu = new Menu("class files");
submenu.add(new MenuItem("TextDemo.class");
submenu.add(new MenuItem("ChoiceDemo.class");
submenu.add(new MenuItem("ScrollbarsDemo.class");
open.add(submenu);
// adds sunmenu to main menu
Mernu exit=new Menu("Exit");
MenuShortcut mshortcut=new MenuShortcut('c');   // Ctrl+c
MenuItem close = new MenuItem("close",mshortcut);
exit.add(close);
exit.addActionListener(this);
```

MenuShortcut.java:-

```
import java.awt.*;
import java.awt.event.*;
publoic class MenuShortcut extends Frame implements ActionListener{
        MenuShortcut(){
                SetLayout(new FlowLayout());
                MenuBar mb=new MenuBar();
                SetMenuBar(mb);
                Menu open=new Menu("open);
                CheckboxMenuItem cb=new CheckboxMenuItem("extended.dat");
                open.add(cb);
                open.add(new MenuItem("line.txt");
                open.add(new MenuItem("readline.txt");
                open.addSeperator();
                submenu.addActionListener(this);
                open.addActionListener(this);
                Menu date=new Menu("Date");
                date.add(new MenuItem("completeInfo");
                date.add(new MenuItem("only year");
                date.add(new MenuItem("only date");
                date.add(new MenuItem("hours and minutes");
                date.addActionListener(this);
                mb.add((open);
                setTitle("sorry, no  actions in my menu");
                setSize(300,300);
                setVisible(true);
                //write the code to close a window
        }
        public void actionPerformed(ActionEvent ae){
                Stirng str=ae.getActionCommand();
                System.out.println("item you clicked is :"+str);
                try{
                        FileInputStream fis=new FileInputStream(str);
                        int k;
                        while((k=fis.read())!=-1)
                                System.out.println(char(k));
                }
                catch(Exception e){
                        System.out.println("exception occured:"+e);
                }
                java.util.Date d=new java.util.Date();
                if(str.equals("completeinfo"))
                        System.out.println(d.toString());
                else if(str.equals("only Date"))
                        System.out.println("today date is :"+d.getDate());
//getxxx() methods of Date class.
                else if(str.equals("hours and minutes"))
                        System.out.println("hours :"+d.getHours() + "munutes
:"+d.getMinutes());
                else if(str.equals("only year");
                        System.out.println("year is :"+d.getYear());
                else if(str.equals("close"))
                        System.exit(0);
        }   //action performed close

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