Event handling :-

Applets are even driven programs. Thus, event handling is at the core of successful applet programming.

Detection Event Model :-

It defines a statement an consistent mechanism to generate and process events. Here the concept is a source generates an event and sense it to one or more listeners. In this scheme the listener simply waits until it receives an event. Once an event is received the listener processes the event and then returns. In this model listener must register with a source in order to receives and event notification.

Event :-

An event is an object that describes a straight chance in a source. E.g. — pressing a button entering a character via keyboard etc. are user defined events. Event may also be generated when a timer expires, a counter exceeds, a software or hardware failure (internal event).

Event source :-

A source is an object that generates an event. This occurs when internal state of that object changes some way. Source may generate more than one type of event.

A source must register listener in order for the listener to receive notification about a specific type of event.

The general syntax for that :-

public void add TypeListner(TypeListner el)

Here type is the name of the event and el is the reference to that listener.

public void add KeyListner(KeyListner kl)

public void add MouseListner(MouseListnermle)

When an event occurs all registered listeners are notified and receives a copy of that event object (this is known as the multicasting the event).

Some sources may allow only one listener to register.

public void add TypeListner(TypeListner el)

throws java.util.TooManyListenerException (This is known as the uni-casting the event)

Event Listener :-

A listeners is an object that is notified when an event occurs. The methods that receives and process events are defined in a set of interfaces found in java.awt.event. E.g.— the MouseMotionListener interface defines method to receive notification when the mouse is dragged or moved.

Event Class :-

The class that represent events are event class. At the root of the java event class hierarchy is "Event Object" which is defined in java.util. Thus it is a super class of all events.

Main Event Classes :-

1. ActionEvent :- Normal action are taken by it. (button press)
2. AdjucentEvent :- Scroll bar.
3. ComponentEvent :- To move all the visible events.
4. ContainerEvent :- For adding or removing things.
5. FocusEvent :- To focus on a particular things. Ex. — Using tab shifting cursor to one button to another.
6. InputEvent :- For taking any input.
7. ItemEvent :- To select or be select a checkbox or listbox or radiobutton.
8. KeyEvent :- For taking input from key board.
9. MouseEvent :- Mouse click/drag/up/down.
10. WindowEvent :- To open or close a frame or a dialog box.

Example :-

```
import java.awt.*;
import java.event.*;
import java.applet.*;
/*<applet code = "Simple" width=300 height=100>
  </applet>*/
class simple extends Applet implements KeyListener {
        String msg=" ";
```

```
        public void init() {
                addKeyListner(this);
                requestFocus();
        }
        public void KeyPressed(KeyEvent ke) {
                showstates("Key up");
        }
        public void KeyTyped(KeyEvent ke) {
                msg t = ke.getKeyChar();
                repaint();
        }
        public void paint(Graphics g) {
                g.drawString(msg,10,20);
}       }
```

AWT (Abstract Windowing Toolkit) :-

The Java programming language class library provides a user interface toolkit called the abstract windowing toolkit. It is use to create GUI based application. A GUI is build of graphical elements called components. Typical components includes buttons scroll bar and text fields. Components allow the user to interact with the program and provide the user with visual feedback about the state of the program. Components do not standalone but rather are found with in containers. Containers contain a control the layout of components. So, at the top of the AWT hierarchy is the component class (it is an abstract class) that encapsulates all the attributes of a visual component. Container class is a subclass of component.

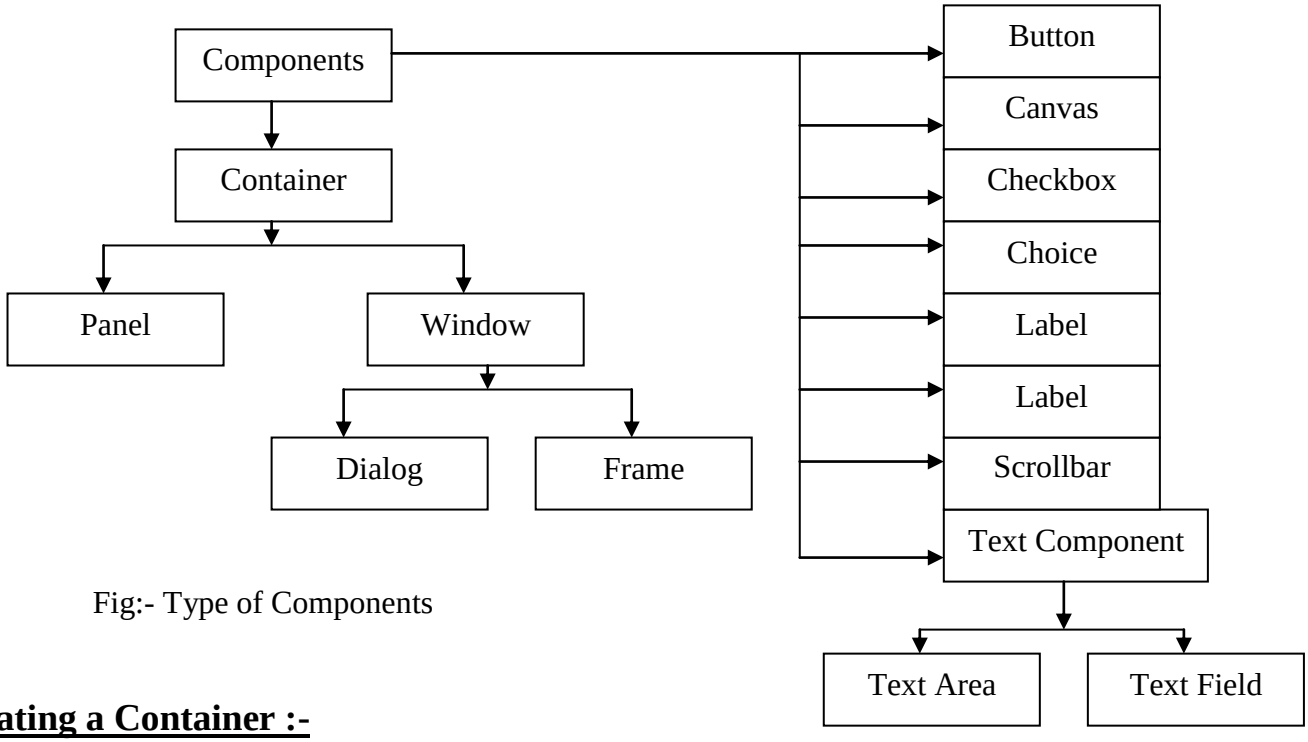

Fig:- Type of Components

Creating a Container :-

Before adding the component that make up a user interface the programmer must create a container. To do so the programmer must create an instance of class window or class frame.

```
/* An empty frame */
import java.awt.*;
public class Example {
        public static void main () {
                Frame f = new Frame("Example");
                f.show();
}       }
import java.awt.*;
public class Example3 extends java.applet.Applet {
        public void init() {
                add(new Button("One"));
```

```
            add(new Button("Two"));
        }
        public Dimension preferredSize() {
            return new Dimension(200,100);
        }
        public static void main() {
            Frame f = new Frame("Example3");
            Example3 ex = new Example3();
            ex.init();
            f.add("Center", ex);
            f.pack();
            f.show();
}       }
```

A frame is initially invisible and must be made visible by invoking its show method.
The pack() sizes the frame so that all its contains are at or above their prefer size.

setSize()
setBound()

```
/* Handling Buttons */
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/* <applet code="ButtonDemo" width=250 height=150>
   </applet> */
public class ButtonDemo extends Applet implements ActionListener {
        String msg = " ";
        Button yes, no, maybe;
        public void init() {
            yes = new Button("Yes");
            no = new Button("No");
            maybe = new Button("Undecided");
            add(yes);
            add(no);
            add(maybe);
            yes.addActionListener(this);
            no.addActionListener(this);
            maybe.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae) {
            String str = ae.getActionCommand();
            If(str.equals("Yes")) {
                msg = "You pressed Yes";
            }
            else if(str.equals("No")) {
                msg = "You pressed No";
            }
            else {
                msg = "You pressed Undefined";
            }
            repaint();
        }
        public void point(Graphics g) {
            g.drawString(msg, 6, 100);
}       }
```