Events :-

Event is an action performed by a user. These actions can be pressing a key, pressing mouse button, clicking mouse, moving mouse etc. When such an action is performed then an event is generated, such events should be handled. The process of handling event is called event handling. There is an approach to event handling called "delegation event model". Here a 'source' generates an event and sends it to one or more 'listeners'. The 'listens' wait until it receives an event. After receiving the event, the listener processes it and returns.

Event sources :-

sources from which events are generated are given below :

1. button - it generates action event when the user presses a button.
2. Checkbox - " item events when the checkbox is selected & deselected.
3. choice - " item events when the user makes a new choice
4. List - it generates action events when an item is double clicked. It also generates item event when item is selected & deselected.
5. Menu Item - it generates both action & item events
6. Scroll bar - it generates adjustment event, when scroll bar is manipulated.
7. Text Components - it generates text events when user enters a character.
8. window - it generates window event, when a window is activated, deactivated, opened & closed.

Event classes :-

EventObject is at the root of the Java event class hierarchy, which is the superclass for all events. Its constructor is shown below -

EventObject (Object Src)

here 'Src' is the object that generates this event.

1. ActionEvent class - this event is generated when a button is pressed, a item is double-clicked or a menu item is selected. This class has four integer constants that can be used to identify any modifiers associated with action event. They are - ALT_MASK, CTRL-MASK, META-MASK and SHIFT-MASK.

This class has 3 constructors :

ActionEvent (Object Src, int type, String cmd)

ActionEvent (Object Src, int type, String cmd, int modifiers)

ActionEvent (Object Src, int type, String cmd, long when, int modifiers)

here 'Src' is a reference to the object that generated this event, 'type' is the type of event, 'cmd' is the command string, 'modifiers' means which modifier keys are pressed, 'when' indicates when the event is occurred.

we can obtain the command name using 'getActionCommand()'

2. AdjustmentEvent class -

it is generated by a scroll bar. It defines 5 constants -

BLOCK-DECREMENT : the user clicked inside the scroll bar to decrease its value

BLOCK-INCREMENT : " " to increase "

TRACK : the slider was dragged.

UNIT-DECREMENT : the button at the end of scroll bar was clicked to decrease its value

UNIT-INCREMENT : " " to increase its value

It has 1 constructor -

AdjustmentEvent (Object Src, int id, int type, int data)

where 'Src' is a reference to the object that generated this event

'id' specifies the event.

'type' is the type of adjustment

'data' is the its associated data.

ComponentEvent class :

This event is generated when the size, position of a Component is changed. There are 4 constants -

COMPONENT- HIDDEN - the Component was hidden
COMPONENT - MOVED - " moved
COMPONENT - RESIZED - " resized
COMPONENT - SHOWN - " visible

It has 1 constructor -

ComponentEvent (Object Src, int type)

where 'type' is the type of event.

4. ContainerEvent class - this event is generated when a component is added to & removed from a container. There are 2 constants -

COMPONENT- ADDED
COMPONENT- REMOVED.

It has 1 constructor -

ContainerEvent (Object Src, int type, Component comp)

5. InputEvent class :

The different Constants are -

ALT_MASK BUTTON1 - MASK BUTTON2 - MASK BUTTON3-MASK
META-MASK SHIFT - MASK CTRL - MASK

To test if a modifier was pressed at the time an event is generated, we use the following methods -

boolean isAltDown ()
" isControlDown ()
" isShiftDown ()

6. ItemEvent class : it is generated when a check box & item is clicked & a menu item is selected & deselected. It has 2 constants -

SELECTED - the user selected an item
DESELECTED - the user deselected an item

It has 1 constructor -

ItemEvent (Object Src, int type, Object entry, int state)

7. KeyEvent class : It is generated when keyboard input occurs.

It has 3 constants -

KEY - PRESSED
KEY - RELEASED
KEY - TYPED

It has one constructor -

KeyEvent (Object Src, int type, long when, int modifiers, int code, char ch)

8. MouseEvent class : There are 8 constants

MOUSE - CLICKED	-	the when clicked the mouse
MOUSE - DRAGGED	-	" dragged "
MOUSE - ENTERED	-	the mouse is entered
MOUSE - EXITED	-	" exited
MOUSE - MOVED	-	" moved
MOUSE - PRESSED	-	" pressed
MOUSE - RELEASED	-	" released
MOUSE - WHEEL	-	the mouse wheel is moved.

It has one Constructor -

MouseEvent (Object Src, int type, long when, int modifiers, int x, int y, int clicks)

where " the coordinates of the mouse are passed in x & y.

9. MouseWheelEvent class : if a mouse has a wheel, it is located between the left & right buttons. wheel is used for Scrolling. It has 2 constants -

WHEEL - BLOCK - SCROLL	-	a page up & page down Scroll event occurred
WHEEL - UNIT - SCROLL	-	a line up & line down " "

It has one constructor -

MouseWheelEvent (Object Src, int type, long when, int modifiers, int x, int y, int clicks, int ScrollAmt)

10. TextEvent Class :

It is generated by text field & text area. It has 1 constant - TEXT - VALUE - CHANGED. It has 1 Constructor -

TextEvent (Object Src, int type)

WindowEvent class - it has constants like -

WINDOW - ACTIVATED - the window was activated
WINDOW - DEACTIVATED - " deactivated
WINDOW - OPENED - " opened
WINDOW - CLOSED - " closed
WINDOW - CLOSING - " to be closed.

It has 1 constructs -

WindowEvent (Object src, int type)

=

Event Listeners :-

A listener is an object which is notified when an event occurs.

1. ActionListener interface - it defines a method which is called when an action event occurs, ie

void actionperformed (ActionEvent ae)

2. AdjustmentListener interface - it defines a method which is called when an adjustment of any field occurs,

void adjustmentValueChanged (AdjustmentEvent ae)

3. ComponentListener - the methods are -

void componentResized (ComponentEvent ce)
void componentMoved (")
void componentShown (")
void componentHidden (")

4. ContainerListener - it defines methods which are called when any Component is added & removed to the container. They are -

void componentAdded (ContainerEvent ce)
void componentRemoved (")

5. ItemListener - method is -

void itemStateChanged (ItemEvent ie)

it is called when the state of an item is changed.

6. KeyListener : methods are -

void keyPressed (KeyEvent ke) - it is called when a key is
void keyReleased (") - it is called when a key is
void keyTyped (") - it is called when a character is typed

7. MouseListener : methods are -

void mouseClicked (MouseEvent me) - it is called when a mouse is clicked
void mouseEntered (") - " " is entered
void mouseExited (") - " " is exited
void mousePressed (") - " " is pressed
void mouseReleased (") - " " is released

8. MouseMotionListener - methods are -

void mouseDragged (MouseEvent me) - it is called when a mouse is dragged
void mouseMoved (") - " " is moved

9. MouseWheelListener - method is -

void mouseWheelMoved (MouseEvent me) - it is called wheel is moved.

10. TextListener - method is -

void textChanged (TextEvent te) - it is called when a text field a text area is modified & changed.

11. WindowListener - methods are -

void windowActivated (WindowEvent we) - it is called when a window is activated
void windowDeactivated (") - " " is deactivated
void windowOpened (") - " " is opened
void windowClosed (") - " " is closed
void windowClosing (") - " " is to be closed

...ation Event model :-

In this model a component generates an event and sends to a listener. The listener immediately receives it & processes it. This is a good design pattern where event handling mechanism code is separated from the user interface Components that generate the events. The advantage is - without affecting the event handling code, we can change the code of GUI with which user interact.

advantages -

1. it gives more performance
2. it is easy to operate
3. we can build more flexible programs

example 1 : MouseEvent handling

```
import java.awt.*;
import java.awt.event.*;
public class one extends Frame implements MouseListener
{
   SetTitle (" handling of mouse event");
   Set Size (100,200);
   SetVisible (true);
   addMouseListener (this);
  public void mouseEntered (MouseEvent me)
  {
      Set Background (Color.red);
  }
  public void mouse Exited (Mouse Event me)
  {
      Set Background (Color. blue);
  }
  pubic void mousepressed (MouseEvent me)
  {
      Set Background (Color. yellow);
  }
  public void mouseReleased (MouseEvent me)
  {
      Set Background (Color. green);
  }
```

```
    public void mouseClicked (MouseEvent me)
    {
        setBackground (Color.black);
    }
    public static void main(String args[])
    {
        new one();
    }
}
```

example 2 :- KeyEvent handling

```
public class two extends Frame implements KeyListener
{
    setTitle ("key event");
    setSize (100,200);
    setVisible (true);
    addKeyListener (this);
    public void keyPressed (KeyEvent ke)
    {
        setBackground (Color.red);
    }
    public void keyReleased (")
    {
        setBackground (Color.yellow);
    }
    public void keyTyped (")
    {
        setBackground (Color.blue);
    }
    public static void main (String args[])
    {
        new two();
    }
}
```

=

pter classes :- Adapter classes

It is time consuming to override all the methods of an interface to handle particular event. For example, to close a window we need to override all the abstract methods of windowListener interface. Even though, we need to override only one method namely windowClosing() we are forced to override all the remaining methods of this interface.

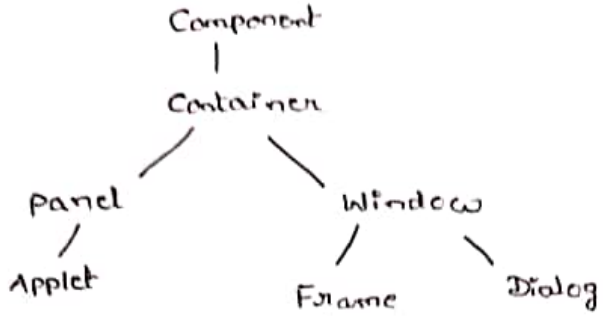
Java provides a solution to this problem by using adapter classes. By this adapter class we need not override all the methods of windowListener interface, we can just override our interested method windowClosing() only by using adapter class provided for WindowListener.

The commonly used adapter classes are -

Event Listener	Adapter class	add() method
WindowListener	WindowAdapter	addWindowListener()
MouseListener	MouseAdapter	addMouseListener()
MouseMotionListener	MouseMotionAdapter	addMouseMotionListener()
ComponentListener	ComponentAdapter	addComponentListener()
ContainerListener	ContainerAdapter	addContainerListener()
KeyListener	KeyAdapter	addKeyListener()

AWT class hierarchy :-

AWT - Abstract Window Toolkit



Component - all user interface elements that are displayed on the screen & that interact with the user are subclasses of Component. It defines methods that are responsible for managing events.

Container - it is subclass of Component. It has additional methods that allow other Component objects to be nested within it. A Container is responsible for layout managers.

Panel - a panel is a window that doesnot contain a title bar, menu bar & border.

Applet - it is a small application that can run on internet browser

Window - it is not contained within any other object, it sits directly on desktop. we cann't create directly 'window' objects. so we use subclass of window called Frame.

Frame - it has title bar, menu bar & border.

Interface Components (or) AWT Components :-

Controls are the Components that allow a user to interact with your application in various ways.

The different Components are —

to add a component in a window add() method is used.

```
Component add ( Component obj);
```

to remove a component from a window remove() method is used.

```
remove (Component obj);
```

The different Components are —

1. Labels :-

A label contains a string, which it displays. Labels do not provide any interaction with the user. It has constructors like —

```
Label ()
Label ( String str )
Label ( String str, int how)
```

first version creates a blank label. The second version creates a label with the string 'str'. The third version creates a label with string 'str' using the alignment specified by 'how'. The values of 'how' are —

Label.LEFT, Label.RIGHT, Label.CENTER.

The methods are :-

a. void setText (String str) — it can set or change the text of a label.
b. getText() — we can read the text of a label.
c. getAlignment() — we can read alignment of a label.
d. setAlignment() — we can change the alignment of a label.

program :-

```
import java.awt.*;
import java.applet.*;
/*
  <applet code = "one" width = 200 height = 200>
  </applet>
*/
```

```
public class one extends Applet
{
    public void init()
    {
        Label l1 = new Label ("one");
        Label l2 = new Label ("Two");
        Label l3 = new Label ("Three");

            add (l1);
            add (l2);
            add (l3);
    }
}
```

2. Buttons :-

A button is a Component that contains a label & that generates an event when it is pressed.

Its Constructors are -

a. Button() - it creates empty button

b. Button (String str) - it creates a button with a label 'str'.

The methods are -

a. setLabel (String str) - it can change the label of a button

b. getLabel() - it reads the label of a button.

program :-

```
public class one extends Applet implements ActionListener
{
    String msg = " ";
    public void init()
    {
        Button b1 = new Button ("C");
        Button b2 = new Button ("C++");
        Button b3 = new Button ("Java");

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

            b1.addActionListener (this);
            b2.          "
            b3.          "
    }
```

```
ublic void actionPerformed (ActionEvent ae)
{
    String str = ae.getActionCommand();
    if (str.equals("C"))
    {
        msg = "u have pressed C";
    }
    else if (str.equals("C++"))
    {
        msg = "u have pressed C++";
    }
    else
    {
        msg = "u have pressed Java";
    }
    repaint();
}
public void paint (Graphics g)
{
    g.drawString (msg, 6, 100);
}
}
```

3. Checkbox :-

A Checkbox is a control that is used to turn an option on or off. It has a small box that can either contain a check mark or not. There is a label associated with each check box that describes what option the box represents.

The constructors are -

a. Checkbox() - it creates a checkbox with empty label, state is unchecked

b. Checkbox(String str) - it creates a Checkbox with a label 'str', state is unchecked

c. Checkbox(String str, boolean on) - if 'on' is true, the check box is initially checked, otherwise it is unchecked.

d. Checkbox(String str, boolean on, CheckboxGroup cbg) - it creates a checkbox with label 'str' & whose group is 'cbg'.

e. Checkbox(String str, CheckboxGroup cbg, boolean on)

The methods are –

a. boolean getState() – to read current state of a check box

b. setState(boolean on) – it changes state of a check box

c. getLabel() – it reads label of a check box

d. setLabel(String str) – it changes the label of a checkbox

Program :-

```
public class one extends Applet implements ItemListener
{
  String msg = " ";
  public void init()
  {
    Checkbox c1 = new Checkbox ("C");
       "     c2 =      "      ("C++");
       "     c3 =      "      ("Java");
        add(c1);
        add(c2);
        add(c3);
    c1. addItemListener(this);
    c2.      "      ;
    c3.      "      ;
}
public void itemStateChanged(ItemEvent ie)
{
  repaint();
}
public void paint(Graphics g)
{
  msg = "Current states are :";
  g.drawString(msg, 6, 80);
  msg = " C :" + c1.getState();
    g.drawString(msg, 6, 100);
  msg = "C++ :" + c2.getState();
      g.drawString(msg, 6, 120);
  msg = "Java" + c3.getState();
      g.drawString(msg, 6, 140);
  }
}
```

O/P

☐ C ☐ C++

☑ Java

Current states are :

C : false

C++ : false

Java : true

ckbox Group :-

It is possible to create a set of check boxes in which only one check box in the group can be checked at any one time. These check boxes are called radio buttons. To create a set of check boxes, you must first define the group to which they will belong & then specify that group when you construct the checkboxes.

The constructs is -

Checkbox Group ( )

The methods are -

a. getSelectedCheckbox ( ) - it determines which checkbox in a group is current selected.

b. setSelectedCheckbox ( ) - we can set a checkbox

5. Choice :-

It is used to create a pop-up list of items from which the user may select. It is a one form of menu. When the user clicks on it, the whole list of choices pops up and a new selection can be made.

The constructs is -

a. Choice ( ) - it creates empty list.

The methods are -

a. add (String str) - it adds a item to the choice

b. getSelectedItem ( ) - it reads the currently selected item

c. int getSelectedIndex ( ) - it reads the index of the currently selected item

d. int getItemCount ( ) - it finds the number of items in the list

e. String getItem (int index) - it finds the item name with the given index

program :-

```
public class one extends Applet implements ItemListener
{
    String msg = " ";
    public void init()
    {
        Choice c1 = new Choice();
        "      c2 = new      "
```

```
c1.add("CDS lab");
c1.add("DBMS lab");
c1.add("Java lab");
c2.add("OOPS");
c2.add("CO");
c2.add("Java");
add(c1);
add(c2);
c1.addItemListener(this);
c2.          "
}
public void itemStateChanged(ItemEvent ie)
{
  repaint();
}
public void paint(Graphics g)
{
  msg = "selected lab =" + c1.getSelectedItem();
      g.drawString(msg, 6, 100);
  msg = "selected theory =" + c2.getSelectedItem();
      g.drawString(msg, 6, 120);
  }
}
```

6. Lists :-

It provides a compact, multiple choice, scrolling selection list. Choice can show only one single selected item in the menu, in a List, it shows any number of choices in a visible window.

The Constructors are -

a. List() - it creates a list that allows only one item to be selected at one tim

b. List(int noofrows) - 'no of rows' specifies the number of entries in the list that will always be visible.

c. List(int no of rows, boolean multipleselect) -

if 'multiple select' = true, then user may select multiple items at a time.

" = false, " only one item at a time.

methods are -

a. void add (String name) - it adds a selection to the list at the end

b. void add (String name, int index) - it adds a selection to the list at the given index value

c. String getSelectedItem() - it reads the currently selected item

d. int getSelectedIndex() - It reads the index of the currently selected item

e. String[] getSelectedItems() - it returns the currently selected multiple items

f. int[] getSelectedIndexes() - it returns the indexes of the currently selected multiple items

g. getItemCount() - it returns the number of selected items.

h. getItem (int index) - it returns the item present at given index.

7. Scrollbars :-

They are used to select continuous values between a specified minimum & maximum. Scrollbars may be oriented horizontally & vertically. There are 2 types of scroll bars -

a. horizontal scroll bar - which has left arrow, slider box, right arrow

b. vertical " - which has up arrow, slider box, down arrow.

The constructors are -

a. Scrollbar() - it creates a vertical scroll bar

b. Scrollbar (int style) - it creates a scroll bar, having orientation of the scroll bar. If 'style' is Scrollbar.VERTICAL, then vertical scroll bar is created. If 'style' is Scrollbar.HORIZONTAL, then horizontal scroll bar is created.

The methods are -

a. int getValue() - it returns the current value of scroll bar

b. void setValue() - it changes the current value of scroll bar.

c. int getMinimum() - it returns the minimum value

d. int getMaximum() - " maximum value

8. Text Field :-

In a text field only one single line text can be entered, it can accept only a few number of characters.

The constructors are -

a. TextField() - it creates a default text field.

b. TextField(int noofchars) - it creates a text field having 'noofchars' size

c. TextField(String str) - it creates a text field with initial value of 'str'

d. TextField(String str, int noofchars) - it creates a text field with initial value of 'str' and having 'no of chars' size.

The methods are -

a. getText() - it reads the text of text field

b. setText(String str) - it sets or changes the text of text field

c. void select(int startindex, int endindex) - it selects the characters from startindex to endindex

d. getSelectedText() - it returns the selected text.

e. void setEchoChar(char ch) - it can disable the echoing of the characters as they are typed. It is used in passwords.

f. getEchoChar() - it reads the echo character.

program :-

```
Public class one extends Applet implements ActionListener
{
   String msg = " ";
   public void init()
   {
     Label l1 = new Label("Name :", Label.LEFT);
     Label l2 = new Label("password :", Label.RIGHT);
     TextField t1 = new TextField(12);
     TextField t2 = new TextField(8);
      t2.SetEchoChar('?');
     add(l1);
     add(t1);                t1.addActionListener(this);
     add(l2);                t2.      "
     add(t2);
   }
```

```
public void actionPerformed (ActionEvent ae)
{ repaint();
}
public void paint (Graphics g)
{
    msg = "Name :" + t1.getText();
        g.drawString (msg, 6, 100);
    msg = "password :" + t2.getText();
        g.drawString (msg, 6, 120);
    }
}
```

9. TextArea :-

Text area can accept multiple lines of text. Initially there is no scrollbars, but after entering some lines of text, then automatically scrollbars will appears.

The constructors are -

a. TextArea() - it creates a default text area.

b. TextArea(String str) - it creates a text area with initial value of 'str'

c. TextArea(int nolines, int nochars) - it creates a text area, with the given 'nolines' & each line can have max. 'nochars'

d. TextArea(String str, int nolines, int nochars) -

e. TextArea(String str, int nolines, int nochars, int sbars) -

where 'sbars' is one of these values -

SCROLLBARS_BOTH SCROLLBARS_NONE

SCROLLBARS_HORIZONTAL_ONLY SCROLLBARS_VARTICAL_ONLY

The methods are -

a. getText() - it returns text of text area

b. setText() - it changes text of text area

c. select() - it selects the characters

d. getSelectedText() - it returns the selected text

e. append(String str) - it appends string 'str' to end of text area

f. insert(String str, int index) - it inserts a string 'str' at the given 'index' value.

Graphics

aphics :-

AWT supports a different types of graphics methods. All graphics are drawn relative to a window. The origin of each window is at the top-left corner and is 0,0. Coordinates are specified in pixels. The Graphics class defines a number of drawing functions. Each shape can be drawn edge-only or filled.

1. drawing lines :-

A line can be drawn by using drawLine() method.

Syntax is -

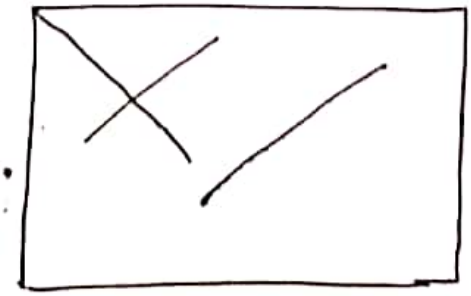
void drawLine (int startX, int startY, int endX, int endY)

it draws a line beginning at startX, startY & ends at endX, endY

program :-

```
import java.awt.*;
import java.applet.*;
/*
  <applet code = "one" width = 100 height = 100>
  </applet>
*/
public class one extends Applet
{
   public void paint (Graphics g)
   {
      g.drawLine (0,0, 100, 100);
      g.drawLine (0,100, 100, 0);
      g.drawLine (40,25, 250,180);
   }
}
```

output



2. Drawing rectangles :-

a. void drawRect (int top, int left, int width, int height)

It displays outlined rectangle.

where the upper left corner of rectangle is at top, left

the dimensions of the rectangle are specified by width & height

b. void fillRect (int top, int left, int width, int height)

it displays filled rectangle.

c. void drawRoundRect (int top, int left, int width, int height, int xdiam, int ydiam)

it displays rounded outlined rectangle.

where xdiam is the diameter of rounding arc along the X-axis

ydiam is " " Y-axis

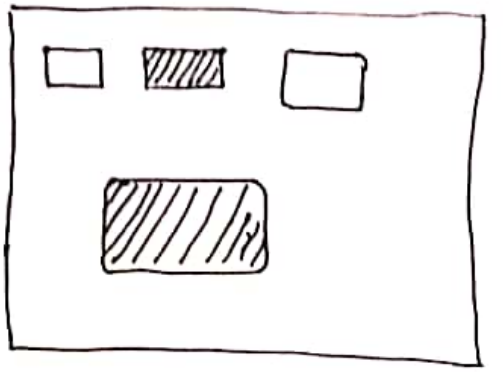
d. void fillRoundRect (int top, int left, int width, int height, int xdiam, int ydiam);

it displays the rounded filled rectangle.

program :-

```
import java.awt.*;
import java.applet.*;
/*
  <applet code = "one"  width = 100  height = 100>
  </applet >
*/
public class one extends Applet
{
   public void paint (Graphics g)
   {
      g.drawRect (10, 10, 60, 50);
      g.fillRect (100, 10, 60, 50);
      g.drawRoundRect (190, 10, 60, 50, 15, 15);
      g.fillRoundRect (70, 90, 140, 100, 30, 40);
   }
}
```

output



by taking same values of width & height we have a square.

drawing . ellipses & circles :-

a. void drawOval (int top, int left, int width, int height);

it displays outlined a ellipse.

b. void fillOval (int top, int left, int width, int height);

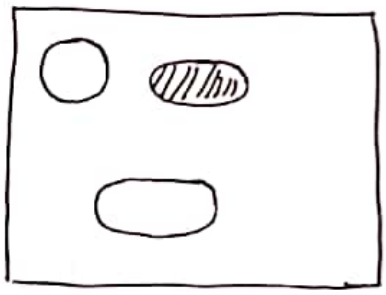
it displays filled ellipse

by taking same values of width & height, we have a circle.

program :-

```
import java.awt.*;
import java.applet.*;
/*
  <applet code = "one"  width = 100  height = 100>
  </applet>
*/
public class one extends Applet
{
  public void paint (Graphics g)
  {
     g.drawOval (10,10,50,50);
     g.fillOval (100,10,75,50);
     g.drawOval (190,10,90,30);
  }
}
```

o/p



4. Drawing arcs :-

a. void drawArc (int top, int left, int width, int height, int startangle, int endangle);

it displays a arc.

where the arc is drawn from startangle through the angular distance specified by endangle. Angles are specified in degrees.

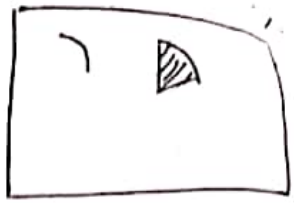
b. void fillArc (int top, int left, int width, int height, int startangle, int endangle)

it displays a filled arc.

program :-

```
import java.awt.*;
import java.applet.*;
/*
  <applet code = "one"  width = 100  height = 100>
  </applet>
*/
```

```
public class one extends Applet
{
    public void paint (Graphics g)
    {
        g.drawArc (10,40,70,70,0,75);
        g.fillArc (100,40,70,70,0,75);
    }
}
```



5. Drawing polygons :-

a. void drawPolygon (int x[], int y[], int numpoints)

it displays a outlined polygon.

the polygon's endpoints are specified by the coordinate pairs contained within the x & y arrays. The number of points defined by x & y is specified by 'numpoints'.

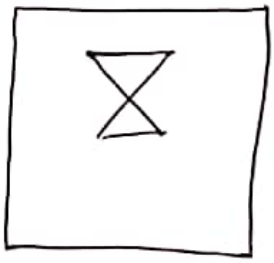
b. void fillPolygon (int x[], int y[], int numpoints)

it displays a filled polygon

program :-

o/p

```
public class one extends Applet
{
    public void paint (Graphics g)
    {
        int a[] = {30,200,30,200,30};
        int b[] = {30,30,200,200,30};
        int n = 5;
        g.drawPolygon (a,b,n);
    }
}
```

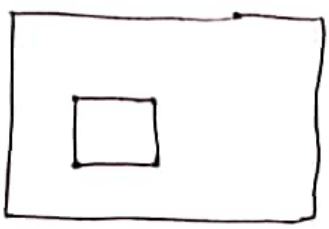


→ by using a set of drawLine() method also we can draw a polygon.

program :-

o/p

```
public class one extends Applet
{
    public void paint (Graphics g)
    {
        g.drawLine (10,10,20,20);
        g.drawLine (20,20,30,30);
        g.drawLine (30,30,40,40);
        g.drawLine (40,40,10,10);
    }
}
```



Layout Managers

ut Managers :-

All the previous AWT components have been positioned by the default layout manager. ie the AWT components in a window are having some default position. But if the user wants to place the components in a window according to his choice, then he can use the layout manager. Layout managers can help the user to place the components according to his choice. There are different layout managers. The required layout manager is selected by setLayout() method. If we have not used this setLayout() method, then the default layout manager is used.

Syntax is -

```
void setLayout (LayoutManager obj)
```

where 'obj' is a reference to the desired layout manager.

If we want to disable the layout manager & position components manually, then pass 'null' for 'obj'.

The different layout managers are -

1. FlowLayout
2. BorderLayout
3. GridLayout
4. CardLayout
5. GridBagLayout

1. FlowLayout :-

It is the default layout manager. It implements a simple layout style, which is similar to how words flow in a text editor. ie in flow layout manager the components are placed starting at upper left corner of the window and then each component is placed from left to right and after completion of one line the remaining components are placed in the next line from left to right. A small gap is present between each component.

The constructors are -

a. FlowLayout() - it creates the default layout ie the components are placed at center & the gap between two components is 5 pixels.

b. FlowLayout (int how) - it specifies how each line is aligned.

The valid values for 'how' are -

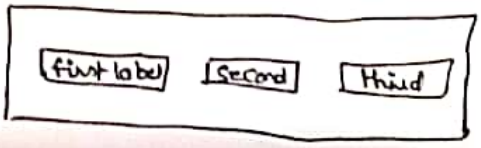
FlowLayout.LEFT → components are at left
FlowLayout.CENTER → " center
FlowLayout.RIGHT → " right
FlowLayout.LEADING → " leading edge
FlowLayout.TRAILING → " trailing edge

c. FlowLayout (int how, int horz, int vert) - it specifies the horizontal and vertical distance between components by horz & vert.

program :-

```
class one extends Frame
{
   one()
   {
      Label l1 = new Label ("first label");
      Label l2 = new Label ("second label");
      Label l3 = new Label ("third label");
      setSize (200, 200);
      setVisible (true);
      setLayout (new FlowLayout());
          add(l1);
          add(l2);
          add(l3);
            show();
      }
      public static void main (String args[])
      {
         new one();
      }
}
```

op



BorderLayout :-

It implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area at the center. The four sides are referred to as north, south, east & west. The middle area is called the center.

The constructors are -

a. BorderLayout() - It creates a default border layout.

b. BorderLayout (int horz, int vert) - it specifies the horizontal & vertical space between components by horz & vert.

BorderLayout defines the following 5 regions -

BorderLayout.CENTER
" . EAST
" . NORTH
" . SOUTH
" . WEST

→ we can add the components to the windows using add() method.

Syntax is -

void add (Component obj, Object region)

where 'obj' is the component to be added

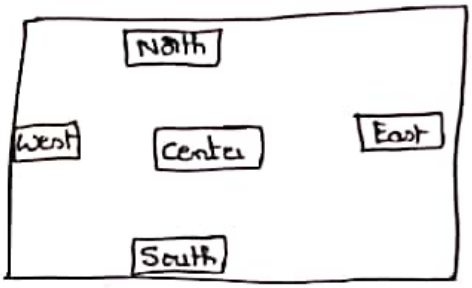
'region' specifies where the component will be added.

program :-

```
class one extends Frame
{
  one()
  {
    Button b1 = new Button ("South");
      "    b2 =    "     ("North");
      "    b3 =    "     ("West");
      "    b4 =    "     ("East");
      "    b5 =    "     ("Center");
    setSize (200,200);
    setVisible (true);
    setLayout (new BorderLayout());
```

```
        add(b1, BorderLayout.SOUTH);
        add(b2, BorderLayout.NORTH);
        add(b3,      "      .WEST);
        add(b4,      "      .EAST);
        add(b5,      "      .CENTER);
        show();
    }
    public static void main(String args[])
    {
        new one();
    }
}
```

o/p



3. GridLayout :-

It places the components in a two-dimensional grid. When you instantiate a GridLayout, you define the number of rows & columns. The container is divided into equal sized rectangles and one component is placed in each rectangle.

The constructors are -

a. GridLayout() - It creates a single column grid layout.

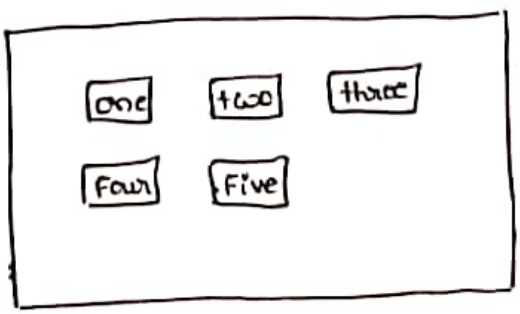
b. GridLayout(int numrows, int numcols) - It creates a grid layout with the specified number of rows & columns.

c. GridLayout(int numrows, int numcols, int horz, int vert) - It specifies the horizontal & vertical space between components.

ogram :-

```
class one extends Frame
{
   one()
   {
      ComponentOrientation c = getComponentOrientation();
      setComponentOrientation (c. RIGHT_TO_LEFT);
      Button b1 = new Button ("one");
        "    b2 =   "    ("Two");
        "    b3 =   "    ("Three");
        "    b4 =   "    ("Four");
        "    b5 =   "    ("Five");
     setLayout (new GridLayout(1));
           add(b1);
           add(b2);
           add(b3);
           add(b4);
           add(b5);
       show();
   }
   public static void main (String args[])
   {
       new one();
    }
}
```

output



4. CardLayout :-

It is a layout manager which treats each component as only one card is visible at a time & the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.

The constructors are -

a. CardLayout () - it creates a default card layout.

b. CardLayout (int horz, int vert) - it specifies the horizontal & vertical distance between the components.

Syntax of add() method is -

```
Void add (Component obj, Object name);
```

where 'name' is a string that specifies the name of the card whose panel is specified by 'obj'.

The different methods are -

a. void first (Container deck) - it retrieves the first card

b. void last (Container deck) - " last card

c. void next (Container deck) - it goes to the next card

d. void previous (Container deck) - it go back to the previous card

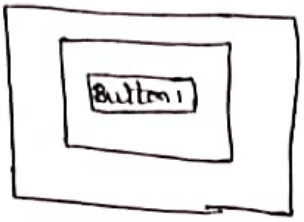
e. void show (Container deck, String cardname) - to see a particular card with the name specified.

program :-

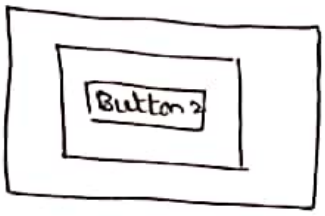
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class one extends JFrame implements ActionListener
{
   Container c;
   CardLayout card;
   JButton b1, b2, b3;
```

```
one().
{
    c = getContentPane();          // create container
    card = new CardLayout(50,10);
    c.setLayout(card);             // set the layout to card layout
    b1 = new JButton("Button 1");
    b2 =     "       ("Button 2");
    b3 =     "       ("Button 3");
    c.add("First card", b1);
    c.add("Second card", b2);
    c.add("Third card", b3);
    b1.addActionListener(this);
    b2.          "          ;
    b3.          "          ;
}
public void actionPerformed(ActionEvent e)
{
    card.next(c);          // shows the next card
    // to show a particular card eg: third card, we use
        card.show(c,"Third Card");     //
}
public static void main(String args[])
{
    one a = new one();
        a.setSize(400,400);
        a.setTitle("Card layout");
        a.setVisible(true);
        a.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

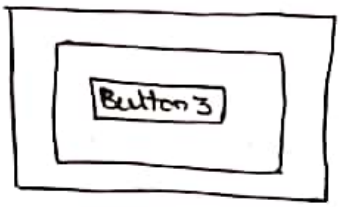
In the above program, the components are arranged whose names are First card, Second card & Third card. When on a card is clicked, the next Card is displayed-



If the Button1 is clicked there, the next card with Button 2 will be displayed as shown below-



If Button 2 is clicked then next card with Button 3 is displayed



5. GridBagLayout :-

Here the components are arranged in rows & columns. This layout is more flexible as compared to other layouts since in this layout, the components can span more than one row or column and the size of the components can be adjusted to fit the display area. The intersection of rows & columns where a component can be placed is called a grid or display area.

When positioning the components by using grid bag layout, it is necessary to apply some constraints or conditions on the components regarding their position, size & space in or around the components etc. Such constraints are specified using GridBagConstraints class.

The constructor is -

GridBagLayout() - it creates a default grid bag layout.

→ to apply some constraints on the components, we should first create an object to GridBagConstraints class. ie

GridBagConstraints c = new GridBagConstraints();

it will create constraints for the components with default values.

→ we can pass some fields to the above class, instead of empty. Some of the fields are -

field	purpose
int anchor	specifies the location of a component within a cell. The default is GridBagConstraints.CENTER
int fill	specifies how a component is resized if the component is smaller than its cell. valid values are - GridBagConstraints.HORIZONTAL " .VERTICAL " .BOTH
int gridheight	specifies height of component in terms of cells
int gridwidth	" width " "
int gridx	" the x-coordinate of the cell to which Component will be added
int gridy	specifies the y-coordinate of the cell to which component will be added
int ipadx	specifies extra horizontal spaces
int ipady	" " vertical "

program :-

```
/*<applet  code = "one"  width = 100  height = 100>
  </applet>
*/
public class one extends Applet implements ItemListener
{
  String msg = " ";
  public void init()
   {
     GridBagLayout g1 = new GridBagLayout();
     GridBagConstraints g2 = new GridBagConstraints();
     setLayout (g1);
     Checkbox c1 = new Checkbox ("C");
       "      c2 =      "     ("C++");
       "      c3 =      "     ("Java");
    // define the grid bag
     g2.weightx = 1.0;
     g2.ipadx = 200;
     g2.insets = new Insets(4,4,0,0);
     g2.anchor = GridBagConstraints.NORTHEAST;
     g2.gridwidth = GridBagConstraints.RELATIVE;
        g1.setConstraints (c1, g2);
     g2.gridwidth = GridBagConstraints.REMAINDER;
        g1.setConstraints (c2, g2);
     g2.gridwidth = GridBagConstraints.RELATIVE;
        g1.setConstraint (c3, g2);
    add(c1);
    add(c2);
    add(c3);
      c1.addItemListena(this);
      c2.      "          ;
      c3.      "          ;
}
```

```
public void itemStateChanged (ItemEvent e)
{
    repaint();
}
public void paint (Graphics g)
{
    msg = "current state :";
    g.drawString (msg, 6, 80);
    msg = " C :" + c1.getState();
    g.drawString (msg, 6, 100);
    msg = "C++ :" + c2.getState();
    g.drawString (msg, 6, 120);
    msg = "Java :" + c3.getState();
    g.drawString (msg, 6, 140);
}
}
```

output