Unit-VI :- AWT

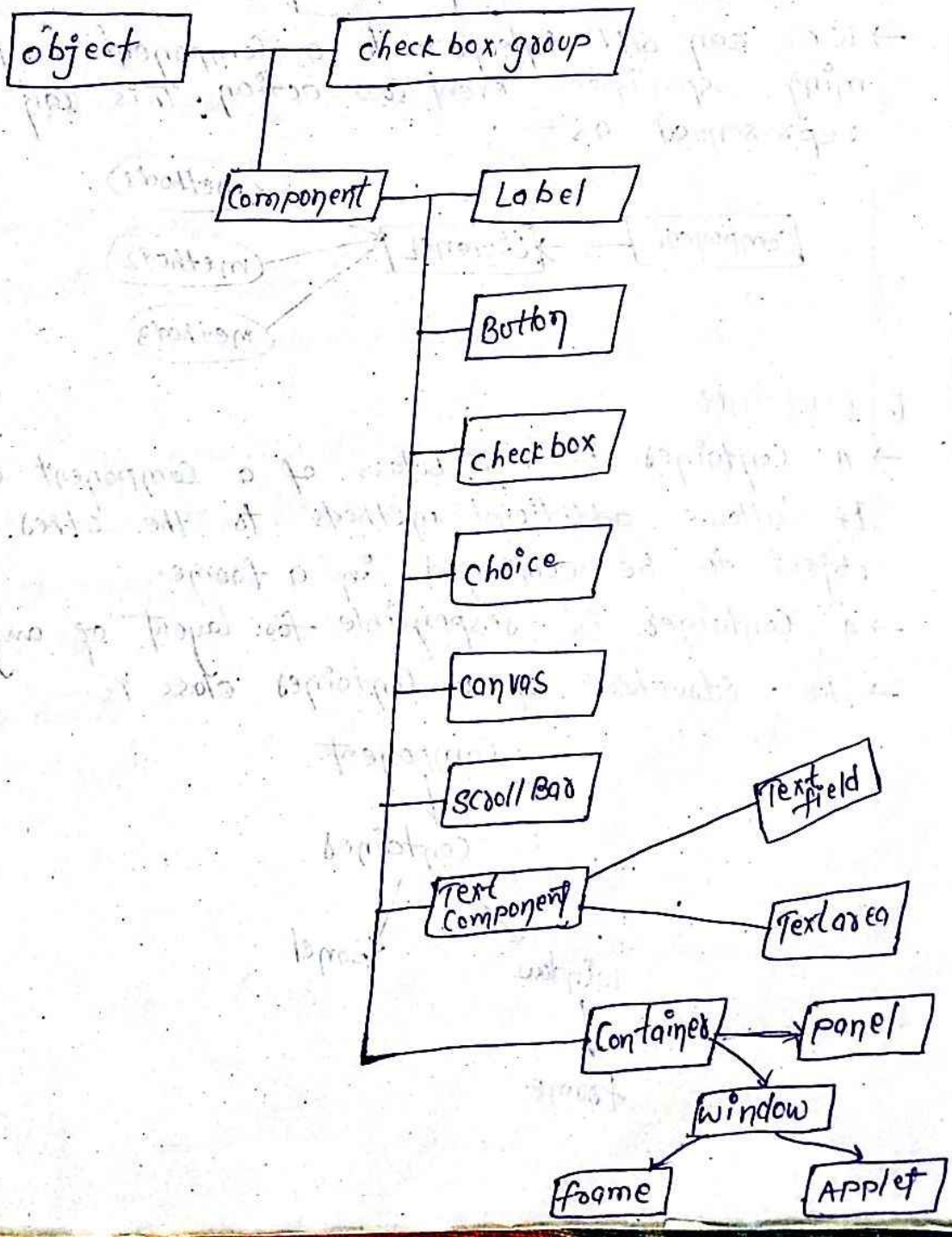
⇒ Abstract window toolkit (AWT) :-

AWT stands for an abstract window toolkit. It represents a class library to develop applications using graphical used interface (GUI).

⇒ Java.awt.package :-

→ This package contains classes and interfaces to develop GUI applications the used can interact easily when the application is developed with the help of graphical used interface.

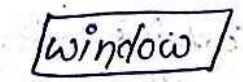
→ The classes of AWT packages are—



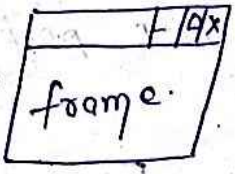
① Component :-

→ A component represents an object which will be display in a window (or) frame. A window represents an imaginary rectangular box which does not have any border (or) title bar.

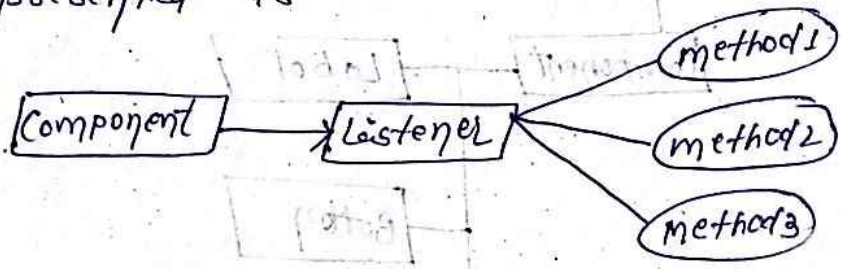
→ This can be represented as —



→ if a rectangular box represents with a border an a title bar is known as frame. This can be represented as—



→ We can add interface to a component for performing specific event (or) action. This can be represented as —

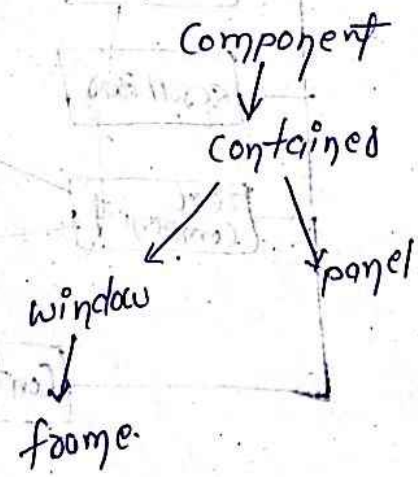


② Container :-

→ A Container is a subclass of a Component class. It allows additional methods to the other Component object to be combined in a frame.

→ A Container is responsible for layout of any Component.

→ The structure of a Container class is —



③ Label :-

→ A label is a constant string that doesn't contains any type of actions.

→ for creating a label we can use the following three forms :-

```
label();
label (string, str);
label (string str, int how);
```

→ In the above three forms the first form create a label without string and the second form creates a label with a string value and the third form also create a label with string value and also we can adjust the label by using LABEL.LEFT, LABEL.RIGHT and LABEL.CENTER.

eg:

```
Import java.awt.*;
Import java.applet.*;
Public class labeldemo extends Applet
{
label one = new label ("yes");
label two = new label ("No");
label three = new label ("may be");
add(one);
add(two);
add(three);
}
}
<html>
<applet code = " labelclemo.class" width=200 height=300>
</applet>
</html>
```

O/P :

⑥ Button:-

→ The Button are useful to perform some action the action for a button can be implemented by using action listener interface.

→ This interface defines action performed method which can be called, when an event (or) an action is generated after clicking the button.

→ for creating a Button we can use the following two statements—

```
b=new Button();
b=new button("label");
```

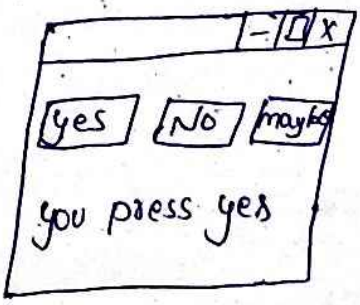
→ in the above forms the first statement is used to create a button without any name. The second statement is used to create a button with a name.

eg:

```
Import Java.awt.*;
import Java.applet.*;
import Java.awt.event.*;
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("may be");
add(yes);
add(No);
add(may be);
yes.addActionListener(this);
```

```
No.addActionListener(this);
maybe.addActionListener(this);
}
Public void Action performed (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 may be";
}
repaint();
}
public void paint(Graphics g)
{
g.drawstring (msg, 60,50);
}
}
<html>
<applet code = "ButtonDemo.class" width=300 height=200>
</applet>
</html>
```

o/p's



⑤ checkbox :-

→ A check box is a control that can be used to turn an option on (or) off it contains a small check-box and a label is associated with each checkbox.

→ for creating a checkbox() we can use the following three statements—

```
checkbox();
checkbox(string str);
checkbox(string str, Boolean on);
```

→ In the above 3-statements the first statements creates a checkbox without any label.

→ The second statement create a checkbox with a label and the third statement we can say the check box option on for displaying the selected checkbox label.

eg:

```
import java.awt.*;
import java.awt.event.*;
import java.awt.applet.*;
Public class checkboxDemo extends Applet
                    Implements itemListener
{
 win = new checkbox ("window 98, true);
 winNT = new checkbox ("windowNT");
 Solaries = new checkbox ("solaries");
 mac = new checkbox ("mac");
 add(win98);
 add(winNT);
 add(Solaries);
 add(mac);
}
```

```
Public void itemstatechanged (ItemEvent e)
{
repaint();
}
Public void paint(Graphics g)
{
msg = "current state";
g.drawstring(msg, 6, 10);
msg = "windows98" + win98.getstate();
g.drawstring (msg, 6, 10);
msg = "winNT" + winNT.getstate();
g.drawstring(msg, 6, 10);
msg = "solaris + solaris.getstate();
g.drawstring (msg, 6, 10);
msg = "mac" + mac.getstate();
g.drawstring (msg, 6, 10);
msg
}
=
<html>
<applet code = "checkboxdemo.class" width = 200 height = 300>
</applet>
</html>
```

%P:-

⑥ Radio Button:-

→ By using radio button we can select only one item from a group of items. A radio button can be created by using a checkbox group class and checkbox class.

→ for creating a radio button first we should create a checkbox group object after that we should pass the checkbox group object to the checkbox class.

→ This can be represented as -

```
checkboxgroup cbg = new checkboxGroup();
checkbox cb = new checkbox("label", cbg, state);
```

→ In the above two statement the state in the second statement represents, if the state is true then the radio button appears is already selected by default.

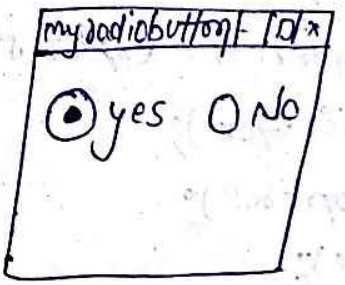
→ if the state is false then the radio button appears as normal means the radiobutton can not be selected.

eg:

```
import java.awt.*;
import java.awt.event.*;
class myradio extends frame Implements ItemListener
{
 String msg = "";
 checkboxGroup cbg;
 checkbox y, n;
 myradio()
 {
 Set layout(new FlowLayout());
 cbg = new checkbox Group();
 y = new checkbox ("yes", cbg, true);
 n = new checkbox ("No", cbg, false);
 add(y);
 add(n);
 y.add ItemListener(this);
```

```
n.addItemListener(this);
Public void itemstatechanged(ItemEvent ie)
{
repaint();
}
Public void paint(Graphics g)
{
 msg="Current selection";
 msg+= cbg.getselected checkbox().getlabel();
 g.drawstring (msg, 10,100);
}
Public static voidmain(string args[])
{
   myradio mr=new myradio();
   mr.set Title ("my radioButton");
   mr.set size(100,400);
   mr.set visible (True);
  }
}
=
<html>
<applet code = "RadioButtonDemo.class" width=200 height=300>
</applet>
</html>
```

O/P:-



⑦ List:-

→ A list contains multiple items for creating a list we can use the following two forms—

```
List();
List(int numRows);
```

→ The first form creates an empty list in the second form the number of rows specified, the number of n-trees that a list may contain.

→ The get selected item and get selected index methods allow the user to select a single item at a time.

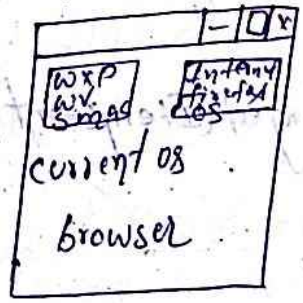
eg:

```
import java.awt.*;
import java.event.*;
import java.applet.*;
public class listdemo extends Applet
                Implements ActionListener
{
  List os, browser;
  String msg=" ";
  Public void init()
  {
  os=new list(4);
  browser =new list(3);
   os.add("window xp");
   os.add("window vista");
   os.add("solaris");
   os.add("mac os");
   browser.add("Internet explorer");
   browser.add("firefox");
   browser.add("opera");
   browser.select(1);
   add(os);
   add(browser);
   os.addActionListener(this);
   browser.addActionListener(this);
}
```

```
Public void actionperformed (ActionEvent ae)
{
repaint();
}
Public void paint(Graphics g)
{
  int idx[];
  msg = "current os";
  idx = os.getselectedIndex();
  for(int i=0; i<idx.length; i++)
  msg+ = os.getItem (idx[i]+" ");
  g.drawstring (msg, 60, 50);
  msg = "current browser";
  msg+ = "browser.getselectedItem();
  g.draw string (msg, 6, 10);
  }
}
```

```
<html>
<applet code = "Listdemo.class" width= 300 height= 200>
</applet>
</html>
```

o/p:



⑧ choice:-

→ A choice is used to create a pop-up list of items that forms a menu.

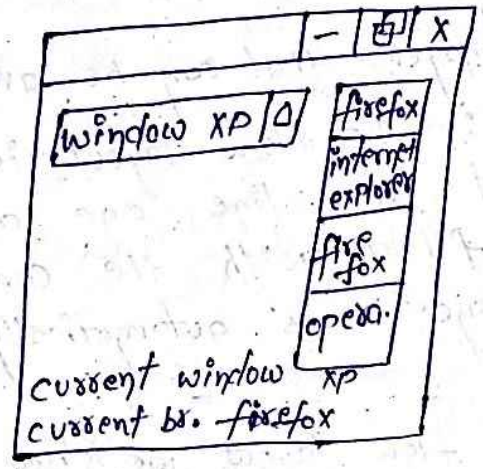
→ whenever the user click on the menu a pop-up list of items will be displayed and the user can select the require.

eg: import java.awt.*;
import java.event.*;

```
import java.awt.*;
Public class choicedemo extends Applet
                    Implements ItemListener
{
 choice os, browser;
 string  msg = " ";
 Public  void init()
 {
  os = new choice (4);
  browser = new choice(3);
  os.add ("window xp");
  os.add ("window vista");
  os.add("solaris");
  os.add("macos");
  browser.add("Internet explorer");
  browser.add("firefox");
  browser.add("opera");
  browser.select(1);
  add(os);
  add(browser);
  os.addItemListener(this);
  browser.addItemListener(this);
 }
 Public void itemstatechanged(ItemEvent ie)
 {
  repaint();
 }
 Public void paint(Graphic g)
 {
  msg = "current os";
  msg+ = os.getselectedItem();
  g.drawstring(msg, 6,120);
  msg = "current browser";
  msg+ = browser.getselectedItem();
```

```
g.drawString(msg, 6, 100);
}
}
*/
<html>
<applet code="ChoiceDemo.class" width=300 height=200>
</applet>
</html>
```

o/p:-



⑨ Contained class:-

A Contained class is a Superclass for all the classes. in a awt. This class has the following methods-

① Container:- This is used to create a new container for adding the component.

② Component add(or) Component cmp:- This is used to add the specified Component at the end of the container.

③ void addContainerListener():- It adds specified Container Listener interface to a Component.

④ void paint(Graphics g):- It is used to paint the Container.

⑤ void add():- This is used to add the specified container.

⑥ void remove(int index):- it is used to remove the specified component from a Container.

⑦ void removeAll():- Remove all the components from a Container.

void removeAll()

→ remove all the components from a container.

void setFont(Font f)

→ It is used to set the font for a container.

void setLayout(LayoutManager mgr)

→ It is used to set the layout manager

void update()

→ It is used to update the container.

Layout managers:

→ A layout manager is a class that can be used to arrange the component in a particular manner. In the frame, the layout managers that can be available in a java are

① Flow layout

⇒ The flow layout is useful to arrange the components in a line one after another when a line is filled with the component the remaining components is automatically placed in the next line. To create a flow layout we can use the following statements.

FlowLayout obj = new FlowLayout();

FlowLayout obj = new FlowLayout(int alignment);

FlowLayout obj = new FlowLayout(int hgap, int vgap);

→ In the above statements the first statement creates a flow layout and the default gap between the component is 5-pixels.

→ In the second statement we specifies alignment to arrange the components from the left. We can use flow layout.LEFT to adjust the component from right side. We can use flow layout.RIGHT and for center. We can use flow layout.CENTER

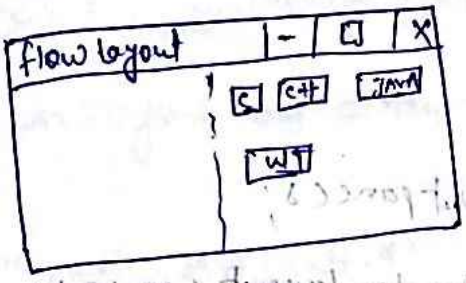
→ In the third statement the hgap and vgap specifies the horizontal and vertical gap between the components.

Ex:-

```
import java.awt.*;
import javax.swing.*;
Class FlowLayout demo extends JFrame
{
FlowLayout demo()
{
container c = get content pane();
FlowLayout obj = new FlowLayout(FlowLayout.RIGHT, 10, 0);
b1 = new JBUTTON("C");
b2 = new JBUTTON("C++");
b3 = new JBUTTON("Java");
b4 = new JBUTTON("INT");
c.add(b1);
c.add(b2);
c.add(b3);
} c.add(b4);
```

```
public static void main(String args[])
{
FlowLayoutDemo demo = new FlowLayoutDemo();
  demo.setSize(400, 400);
  demo.setTitle("FlowLayout");
  demo.setVisible("True");
}
}
```

A. output



Border layout

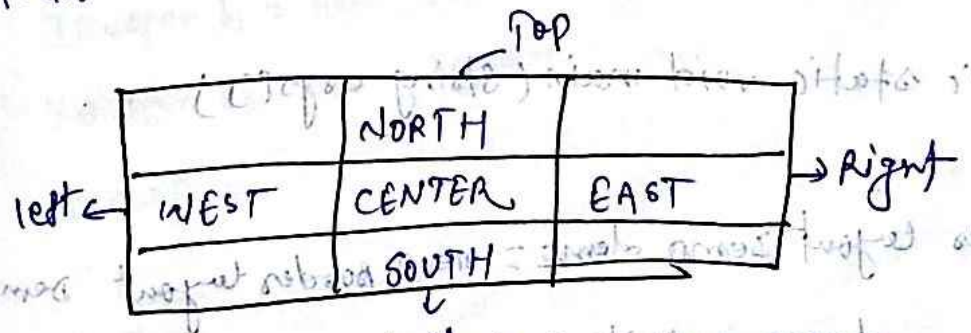
→ Border layout is used to arrange the components is four border in a frame along with center. The border are identify with the names of direction.

→ The top border is specified as north.

→ The right side border is specified as East.

→ The left side border is specified as west.

→ The bottom of the border is specified as south and the middle of the border specified as center this can be represented as



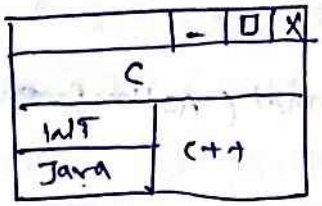
→ for creating a border layout we can use the following two statements:

```
Border layout obj = new Border layout ();
Border layout obj = new Border layout (int hgap, int vgap);
```

Ex!:

```
import Java.awt.*;
import javax.swing.*;
class Borderlayout Demo extends JFrame
{
Border layout demo()
{
container c = get contentpane();
Border layout obj = new Border layout (10,10);
c.setLayout (obj);
JButton b1 = new JButton ("c");
JButton b2 = new JButton ("c++");
JButton b3 = new JButton ("Java");
JButton b4 = new JButton ("DOT");
c.add (b1, Borderlayout.NORTH);
c.add (b2, Borderlayout.EAST);
c.add (b3, Borderlayout.SOUTH);
c.add (b4, Borderlayout.CENTER);
}
public static void main (String args[])
{
Border layout Demo demo = new Border layout Demo();
demo.setSize (400, 400);
demo.visible (true);
```

output



Card layout:

→ In card layout only one component ~~or~~ one card is visible at a time to create a card layout we can uses the following statements

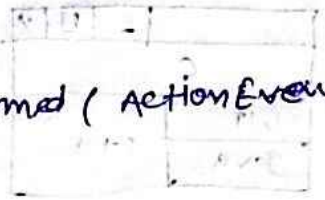
```
CardLayout obj = new CardLayout();
```

Ex:

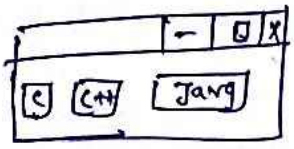
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class cardlayout extends JFrame implements ActionListener
{
    Container c;
    CardLayout card;
    JButton b1, b2, b3;
    card = new CardLayout();
    c.setLayout(card);
    JButton b1 = new JButton("C");
    JButton b2 = new JButton("C++");
    JButton b3 = new JButton("Java");
    c.add("first card", b1);
    c.add("second card", b2);
    c.add("third card", b3);
    b1.addActionListener(this);
```

```
b2.addActionListener(this);
b3.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
    card.next();
}
public static void main(String args[])
{
    CardLayoutDemo demo = new CardLayoutDemo();
    demo.setSize(400, 400);
    demo.setVisible(true);
}
}
```



output



Grid layout

→ A Grid layout is useful to divide the container into a two dimensional grid form that contains several rows and columns.

→ The container is divided into a equal size rectangle and one component is placed in each rectangle to create a grid layout.

→ To create a grid layout, we can use the following statement

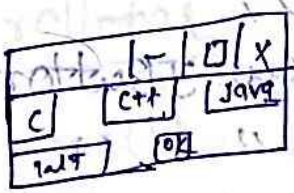
```
Grid layout obj = new GridLayout();
Grid layout obj = new GridLayout(int rows, int columns);
```

Ex:

```
import java.awt.*;
import javax.swing.*;
class GridLayoutDemo extends JFrame
{
```

```
GridLayout Demo()
{
GridLayout grid = new gridLayout (2, 3, 50, 50);
c.setLayout (grid);
JButton b1 = new JButton ("C");
JButton b2 = new JButton ("C++");
JButton b3 = new JButton ("Java");
JButton b4 = new JButton ("JDT");
JButton b5 = new JButton ("DOS");
c.add (b1);
c.add (b2);
c.add (b3);
c.add (b4);
c.add (b5);
}
public static void main (String args[])
{
GridLayout Demo demo = new GridLayout Demo();
demo.setSize (500, 500);
demo.setVisible (true);
}
}
```

o/p:-



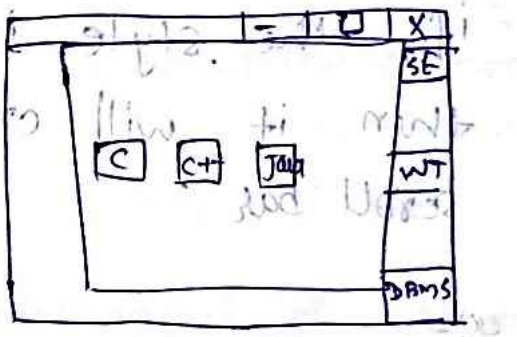
BoxLayout :-

→ A boxLayout allows multiple components to be added either vertically or horizontally to create a box layout. We can use the following statement

```
BoxLayout box = new BoxLayout (JPanel, object, axis,
                               orientation);
```

```
import java.awt.*;
import javax.swing.*;
class BoxLayoutDemo extends JFrame
{
    BoxLayoutDemo()
    {
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        Mypanel mp1 = new Mypanel();
        c.add(mp1);
        Mypanel mp2 = new Mypanel();
        c.add(mp2);
    }
    class Mypanel1 extends Jpanel
    {
        Mypanel1()
        {
            Boxlayout box1 = new BoxLayout(this,
                                BoxLayout.X_AXIS);
            setlayout(box1);
            JButton b1 = new JButton("A");
                 "    b2 =   "    ("B");
                 "    b3 =   "    ("Java");
        }
    }
    class Mypanel2 extends Jpanel
    {
        Mypanel2()
        {
            BoxLayout box2 = new BoxLayout(this
```

```
BoxLayout.Y_AXIS);
JButton b1 = new JButton("SE");
"    b2 = "    "    ("WT");
"    b3 = "    "    ("DBMS");
}
public static void main(String args[])
{
BoxLayoutDemo demo = new BoxLayout
demo.setSize(400,400);
demo.setVisible("true");
}
```



scroll bar

→ A scrollbar is used to select continue values between the minimum and maximum.

→ A scrollbar can be created either vertically or horizontally.

→ To create a scrollbar we can use the following statements.

→ scrollBar();
scrollBar(int style);

→ In the first statement we are not defined the style of scroll bar. It will create a vertical scroll bar by default.

→ The second form allows you to specify the orientation of a scroll bar.

→ If the style is scrollBar.VERTICAL, it will creates a vertical scroll bar, if the style is scrollBar.Horizontal then it will create the horizontal scroll bar.

ex:-

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
class SBDemo extends Applet Implements
                    AdjustmentListener, mouse
                    motionListener
{
    String msg = "";
    ScrollBar vertsb, horzsb;
    public void init()
    {
```

```
int width = Integer.parseInt(getParameter
                              (width));
int height = "  "  "  "  (getParameter
vertsb = new scrollBar(scrollBar.VERTICAL,
                                    (height));
                                0,1,0, height);
horzsb = "  "  "  (scrollBar.HORIZONTAL,
                                0,1,0, width);
add(vertsb);
add(horzsb);
vertsb
vertsb.addAdjustmentListener(this);
horsb.add  "  "  (this);
}
public void adjustment value changed
                        (AdjustmentEvent ae)
{
  repaint();
}
public void mouseDragged(MouseEvent me)
{
  int x = me.getX();
  int y = me.getY();
  vertsb = setvalue(y);
  horzsb = "  "  (x);
  repaint();
}
public void mouseMoved(MouseEvent me)
{
}
```

```
Public void paint (Graphics g)
{
String msg = "vertical:" +vertsb.getValue();
msg = "horizontal:" +horzsb. ");
g.drawString("#", horzsb.getValue(), vertsb.
                                    getValue();
}
}
```

~~Menu~~

MENU :

→ In a menu we can arrange a list of items. for arranging a list of items in a menu, first we have to create a menu bar after creating the menu bar, we have to create the main menu for creating a menu and a menu bar.

→ We have to use the following statements

```
MenuBar mb = new MenuBar();
Menu me = new Menu();
```

EX:-

```
import java.awt.*;
import java.awt.event.*;
public class SimpleMenu extends Jframe implements
                                ActionListener
{
  Menue states, cities;
    public SimpleMenu()
    {
      MenuBar mb= new menuBar();
      SetMenuBar(mb);
        States = new Menu("Indian states");
        cities = "     "("Indian cities");
        mb.add(states);
         mb.add(cities);
        State.add ActionListener(this);
        cities.add ActionListener(this);
        state.add(new MenuItem("Himachal pradesh"));
        state.add(new MenuItem("Rajasthan"));
        State.add(new MenuItem("west Bengal"));
        cities.add(new MenuItem("Delhi"));
          "    "("Jaipur");
          "    "("kolkata"));
      }
      public void action performed(ActionEvent
                                    ae)
      {
        string str = ae.getActionCommand();
        system.out.println("you selected"+str);
      }
```

```
public static void main (String args[])
{
    simpleMenu m1 = new simpleMenu();
    m1.setSize(300,300);
    m1.setVisible(true);
}
}
```

O/P:-