

CHAPTER 6 : APPLET, AWT, Event Handling

It is a small java program which will execute on a web browser. The browser will understand only the HTML tag. Therefore the applet's .class file will not be recognized by the browser. A new tag is created for the browser called <applet>, so that the .class file of the applet will be recognized by the browser. Even if the browser will understand the applet tag, still it is unable to execute the applet because it is a java program, hence each browser contains a reduced version of JVM, when the browser will encounter the <applet> tag of the html file, it will ask the JVM to execute the applet .class file on the browser's window.

ADVANTAGES :-

Compile once, run anywhere : Once will create .class of applet, it will execute on any machine consisting of a browser. No need to recompile it. Hence it enhances the platform independence of java.

No virus Issue : The applet's are not allowed to access the client's memory illegally.

No documentation issue : The documentation of applet can be given within the HTML file. Hence separate documentation is not required.

HOW TO CREATE AN APPLET :- A java program in which we like to provide applet functionality must inherit from Applet class available in java.applet package, and the class must remain as public.

<pre>import java.applet.*; import java.awt.*; public class First extends Applet { public void paint(Graphics g){ g.drawString("hello",75,100); } };</pre> <p>Save the above program as First.java & compile it.</p>	<pre><html><body bgcolor="cyan"> <applet code="First" width="200" height="200"> </applet></body></html></pre> <p>The following source file should be saved as Test.html in the same location where First.class is stored.</p>
---	---

The java.applet package contains Applet class and three interfaces as AppletContext, AudioClip, AppletStub.

Attributes of <applet> Tag:

code	: specifies the applet class file
[codebase]	: is an optional attribute that specifies the base URL of the Applet.
height	: specifies the height of applet
width	: specifies width of applet
[align]	: specifies alignment, the align must constant are LEFT, RIGHT, TEXTTOP, MIDDLE, ABSMIDDLE, BOTTOM, ABSBOTTOM, BASELINE
[vspace]	: specifies vertical space
[Hspace]	: specifies horizontal space
[archive]	: specifies the jar file
[Name]	: specifies the alias name for the applet used for inter applet communication.

Difference between Application & Applet :

Application	Applet
Are independent of OS	Dependent on browser
Are executed under the local o/s.	Executed remotely by a browser.
CUI or GUI	GUI
Execution starts with main()	Execution Starts with init() method.
Terminates with main()	Terminates only when the page is closed

appletviewer : It is an exe file, which is used to execute the applet. It will understand only <applet> or tag (it is similar to reduced version of JVM available in browser) & it will discard rest of the code of a file. If we

write the <applet> within the java source file between comment line, then we can execute the applet by appletviewer from the dos prompt.


dos prompt > appletviewer First.java

The **Graphics** class is an abstract class that represents the display area of an applet. This class is a part of the **java.awt** package and you use it to draw images within the display area of the applet. The object of the Graphics class is used for painting the applet.

Methods of Graphics Class

- drawLine (int x1, int y1, int x2, int y2)
- drawRect (int x, int y, int width, int height)
- drawRoundRect(int x, int y, int w, int h, int acrw, int arch)
- drawstring(String str, int x, int y)
- drawOval(int x, int y, int width, int height)
- drawArch(int x, int y, int w, int h, int startangle, int sweepangle)
- drawPolygon(int x [], int y [], int n)
- drawPolyLine(int x [], int y [], int n [])
- fillRect(int x, int y, int width, int height)
- fillRoundRect(int x, int y, int w, int h, int acrw, int arch)
- fillOval(int x, int y, int width, int height)
- fillArch(int x, int y, int w, int h, int startangle, int sweepangle)
- fillPolygon(int x [], int y [], int n)
- draw3DRect(int x, int y, int width, int height, boolean raised)
- fill3DRect(int x, int y, int width, int height, Boolean raised)
- clearRect(int x, int y, int width, int height)
- setColor(Color c)
- setFont(Font f)
- getColor()
- drawImage(Image img, int x, int y, ImageObserver obj)

LIFE CYCLE OF AN APPLET :- The following methods are known as life cycle method. All the following methods are defined in applet class, except the paint() which is defined in Container class.

When applet is started the following methods are executed sequentially : <pre> public void init() public void start() public void paint(Graphics g) </pre>	
When applet is stopped the following methods are executed sequentially: <pre> public void stop() public void destroy() </pre>	

init() : This method initializes an applet as constructor initialises an object. As the program executions starts from main (), similarly an Applet execution starts from init(). This method will execute when Applet is loaded into the browser for the first time or when user clicks on the refresh button. From this method, the start () is always invoked implicitly.

start() : This method is used to allocate as well as hold the resources necessary to execute the applet. It will execute after init() or when the browser is maximised or when the user revisits the html page consisting of applet. From this method, the paint is invoked implicitly.

stop() : This method releases the resources temporarily hold by the applet. It is invoked before the destroy () when the browser is minimised or when user leaves the html page consisting of applet.

destroy() : This method is used to terminate the execution of an applet. It will execute only once during the life cycle of an applet only when the browser is closed or when user changes to other website.

paint() : This method is used to draw shapes, images, strings etc. on the applet. It is also used to paint the background or foreground colour of the applet. It is always invoked from the start () or when applet will get the focus. The paint () can be invoked explicitly by repaint().

Example : Life cycle

```
import java.applet.*;
import java.awt.*;
/*<applet code="AppLife" height=200 width=300></applet>*/
public class AppLife extends Applet {
    public void init() {
        setBackground(Color.cyan);      setFont(new Font("asdf", Font.BOLD+Font.ITALIC, 40));
        System.out.println("initializing.....");    }
    public void start(){
        System.out.println("starting.....");    }
    public void paint(Graphics g){
        System.out.println("painting.....");    g.drawString("welcome", 50, 100);    }
    public void stop(){
        System.out.println("stopping.....");    }
    public void destroy(){
        System.out.println("destroying.....");    }
};
```

//Creating a banner

```
import java.applet.*;
import java.awt.*;
/*<applet code="AppLife3" width="800" height="200"></applet>*/
public class AppLife3 extends Applet {
    int x=0;
    public void init(){
        setFont(new Font("sadfgas", Font.BOLD, 40));
        setBackground(Color.orange);
        System.out.println("init..");
    }
    public void paint(Graphics g){
        g.drawString("Rama", x ,100);
        if(x==1200) x=0;
        else x=x+10;
        try{
            Thread.sleep(200);
        }catch(Exception e) { }
        repaint();
    }
};
```

AWT PACKAGE(Abstract Window Toolkit) :- This package provides all the GUI (Graphical User Interface) component used in a Java program. The awt hierarchy are as follows:

Component :- **Component** is the base class of awt hierarchy from which all other classes are derived. It is also an abstract class, it encapsulates all the functionalities of an awt component (common methods for GUI components). A **component** is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Components can be resized or repositioned by using methods setLocation(), setSize() or setBounds(), or otherwise arrangement of components will be taken care by the DefaultLayout Manager of the corresponding container.

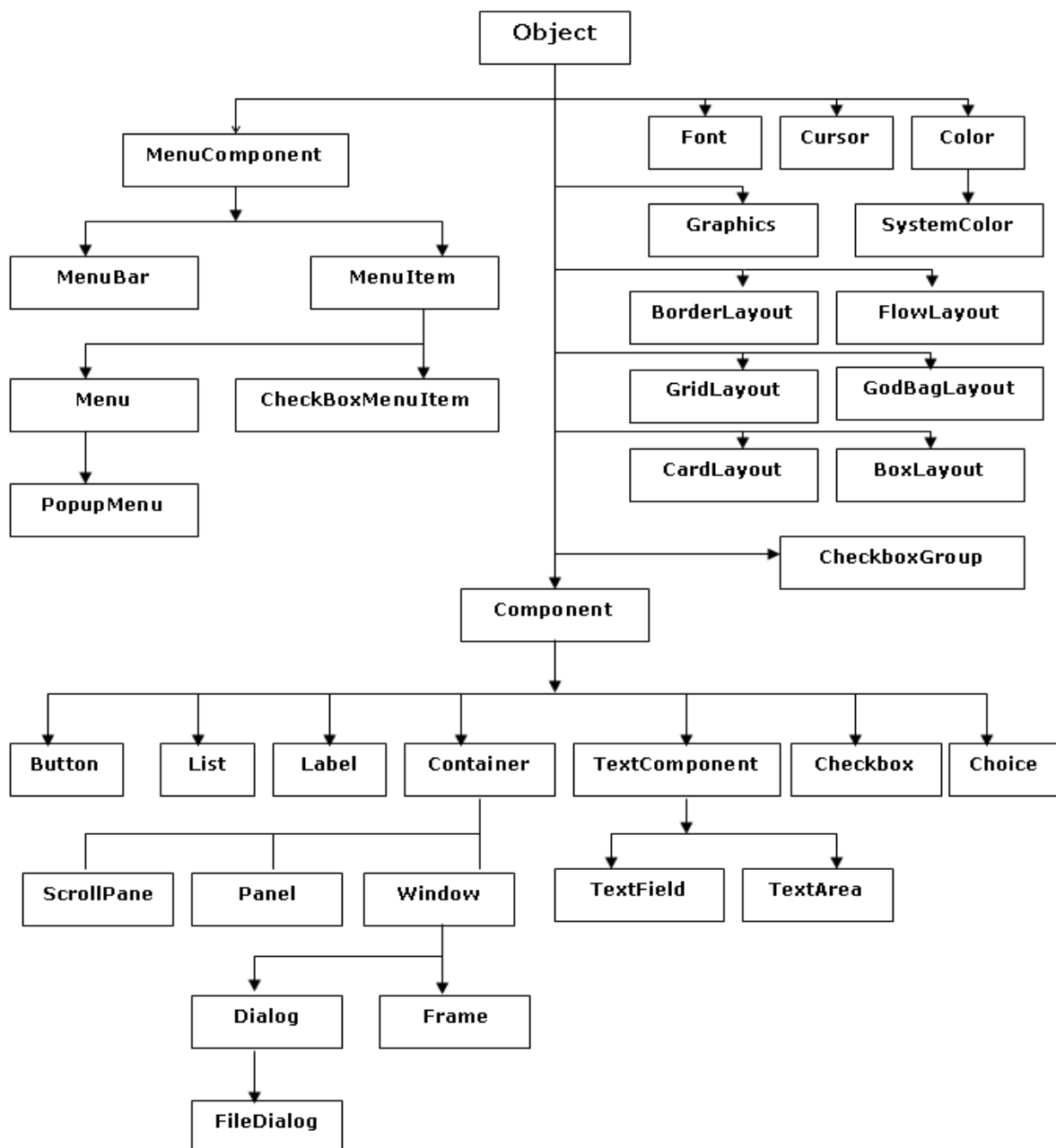
Container :- It is an abstract class like component class. It provides a rectangular area. A container is like a bag where as a component is like a book. Therefore a container can contain some components as well as other containers, but a component can't contain a container.

Panel :- It is a container for internet based application. When applet is visible on browser, first of all a panel is drawn, above it applet is displayed. A panel has no title bar, menu bar or border.

Applet :- This class allows us to create a container for internet based java application by inheriting from it. Applet has no title bar, menu bar or border. But it uses the title bar and menu bar of the browser.

Window :- It is a container for stand alone java application or desktop based GUI or window based application. A window is a top level window that is directly displayed on a desktop. It has no title bar, menu bar or border.

Frame :- This class allows us to create standalone java application (container for desktop based GUI) by inheriting from it. A frame has a title bar or border without the menu bar. It has all the window functionality like minimize, maximize, but has a closing button (with no functionality).



Creating a Frame :

Steps to create Frame:

- 1) The user defined class must inherit a Frame class of awt package instead of Applet class.
- 2) The init() of the applet behaves like constructor which initializes the Applet and main() from where the execution of the Applet is started, hence the init() must be replaced by constructor which will contain the code of init() as well as by main() of the Frame class.
- 3) The main() should create an object of the user defined class that inherits from the Frame class, so that the Frame will be created.
- 4) Since the Frame will be executed by the same interpreter which is used to execute CUI based programs, hence the interpreter will not open the .class file of Frame to see whether it has a GUI interface or not. Therefore, we must use setVisible() by providing true in the argument, so that the Frame will be visible on the desktop/window.
- 5) The size of the applet is generally given in the applet tag, but in case Frame we must specify the size by setSize() method that accepts width & height of the Frame.
- 6) The class that inherits from Frame class must be compiled and executed like a simple java program;

//Example: Frame demo

```
import java.awt.*;
public class FrameDemo extends Frame {
    public FrameDemo() {
        super("Ram");
        Button b1 = new Button("Ok");
        add(b1);
        setSize(400,250);
        setVisible(true);
        setBackground(Color.cyan);
        b1.setBackground(Color.yellow);
        setFont(new Font("asdf", Font.BOLD +Font.ITALIC, 40));
        setLocation(100,150);
        setTitle("Sam");
    }
    public static void main(String rgs[]) {
        new FrameDemo();
    }
};
```

Font class :

Fields : bold, italic, plain

Constructors

- **Font(String name, int style, int size)** → creates a new font with specified name style and size.

Methods:

- public String getName() → Returns the logical name of the font.
- public int getStyle() → Returns the style of the font.
- public int getSize()→ Returns the size of the font.
- public boolean isBold()→ Returns true if the font is BOLD.
- public boolean isItalic()→ Returns true if the font is ITALIC.

Example : Font demo

<pre>import java.awt.*; class FontDemo extends Frame{ Label label; Font font; FontDemo(){ super("FontDemo"); label= new Label("Welcome to Java Font"); setLayout(new FlowLayout()); } // Font Setting for Output-2 //font=new Font("Arial",Font.BOLD,20);</pre>	<pre>// Font Setting for Output-3 font=new Font("Arial",Font.BOLD+Font.ITALIC,40); label.setFont(font); add(label); pack(); setVisible(true); } public static void main(String args[]){ new FontDemo(); }</pre>
---	--

Button : It is a component which allows to click on it.

Button()—Creates an empty Button without any label.

Button(String str) – Creates a Button with the given string as a label.

TextField : It is a component which allows to write data from keyboard in one row.

TextField() – Creates a blank TextField

TextField(int x) – Creates a TextField of the specified columns.

TextField(String str) – Creates TextField which shows the str when the the TextField is displayed.

TextField(String str, int x) – Creates a TextField which displays the text and length will be x.

public void setEchoChar(char c) – sets the character given as the password character.

TextArea : It is a component which allows us write text from keyboard in multiple rows & columns.

TextArea() -- Creates a TextArea which displays the text in mutiple rows.

TextArea(int rows, int cols) – Creates a TextArea by the specified rows & columns.

TextArea(String str) – Creates a TextArea which display the given the text.

TextArea(String str,int rows, int cols) – Creates a TextArea by the specified rows & columns, the str will remain as the text of the TextArea.

All the above TextArea will provide the Vertica & horizontal Scrollbars, whenever necessary.

public void append(String str) – Takes a text and adds the text at the end of previous text.

public String getText() – Returns the text of the **TextArea/TextField/Label**.

public void setText(String str) – Takes a string and keeps the String as the text of the **TextArea/TextField/Label**, and removes the previous text.

Label : It is a component used to show a text/label to the user

Label()—Creates a Label without any caption.

Label(String str) – Creates a Label with the given string as a caption.

Label(String str, int orientation) – Creates a Label with the given string as a caption, and the orientation may be Label.LEFT(it is the default orientation), Label.RIGHT, Label.CENTER

Checkbox : This component allows us to select or deselect an option. It allows multiple selection among the bunch of Checkboxes.

Checkbox() – Creates a Checkbox without any Label

Checkbox(String str) – Creates a Checkbox with str as Label.

Checkbox(String Label, Boolean state) – Creates a Checkbox with a Label, if the 2nd arg. Is true then it will remain selected, the previous 2 constructor will create Checkbox without the selection.

Checkbox(String Label, ChrckboxGroup cg, Boolean state) – Creates a Radiobutton

public boolean getState() – Returns the state of the Checkbox, true –selected, false–deselected.

public void setState(boolean on) – Sets the state of the Checkbox

public String getLabel() – Returns the label of the **Label/Checkbox/Button**.

public void setLabel(String str) – Sets the str as the label of the **Label/Checkbox/Button**.

```
import java.awt.*;
import java.applet.*;
/*<applet code="Comp" width="800"
height="450"></applet> */
public class Comp extends Applet {
    boolean flag ;
    Button b1, b2;
    Label a1, a2;
    TextField tf1, tf2 ;
    TextArea ta;
    public void init(){
        setFont(new Font("adsf", Font.BOLD,30));
        setBackground(Color.cyan);
        b1 = new Button( "Ok" );
        b2 = new Button();
        a1 = new Label( "Username" );
        a2 = new Label();
        tf1 = new TextField( "Ram chandra" , 15 );
        tf2 = new TextField(15 );
```

```
tf2.setEchoChar('*' );
ta = new TextArea(5, 10 );
setComp(a1, Color.pink );
setComp(a2, Color.pink );
setComp(b1, Color.green );
setComp(b2, Color.green );
setComp(tf1, Color.yellow );
setComp(tf2, Color.yellow );
setComp(ta, Color.red );
}
void setComp( Component c, Color z ) {
    c.setBackground( z );
    add( c );
}
public void stop() {
    b2.setLabel( "Cancel" );
    a2.setText( "Password" );
    System.out.println(b1.getLabel());
}
};
```

Choice : This component provides a popup menu of item & allows a single selection of items.

Choice() – Creates a Choice object

public void insert(String item, int index) – Inserts an item into the Choice at the specified index.

public int getSelectedIndex() – Returns the index of currently selected item of the Choice.

List : This component provides a list and allows multiple selection of items.

List() – Creates List object with a space to display 4 items.

List(int rows) – Creates List object with a space to display rows no. of items.

List(int rows, boolean flag) – Creates List object with a space to display rows no. of items, and this allows multiple selection when 2nd arg. becomes true.

public void add(String item, int index) – Adds an element at the index position.

public String[] getSelectedItems() – Returns an array of elements currently selected.

public String getItem[] (int index) – Returns the items in the List.

Common methods of Choice & List are given below :

public String getItem(int index) – Returns an element at the specified index.

public String getSelectedItem() – Returns an element currently selected.

public int getSelectedIndex() – Returns the index of the item currently selected.

public int getItemCount() – Returns the no. of elements available.

public void add(String item) – Adds an element to the end of other items.

public void remove(int index) – Removes an element at the specified index.

public void remove(String item) – Removes an element as specified in parameter.

public void removeAll() – Removes all elements.

public void select(int index) – Selects an element from the List at the specified index.

<pre>import java.applet.*; import java.awt.*; /*<applet code="Chk" width=700 height=200></applet> */ public class Chk extends Applet{ public void init() { setFont(new Font("asdf", Font.BOLD, 40)); setBackground(Color.cyan); //Checkbox Checkbox c1 = new Checkbox("Java", true); Checkbox c2 = new Checkbox("Oracle"); Checkbox c3 = new Checkbox(".Net",true); Checkbox c4 = new Checkbox("Sap"); add(c1); add(c2); add(c3); add(c4); //Radio button CheckboxGroup cg = new CheckboxGroup(); Checkbox r1= new Checkbox("Java", cg, true);</pre>	<pre>Checkbox r2= new Checkbox("Oracle",cg, false); Checkbox r3 = new Checkbox(".Net",cg, true); Checkbox r4 = new Checkbox("Sap", cg, false); add(r1); add(r2); add(r3); add(r4); //Choice Choice c = new Choice(); c.add("aaaa"); c.add("bbbb"); c.add("cccc"); c.add("dddd"); add(c); // List List tt= new List(5, true); for(int i=65 ; i< 75 ;i++) tt.add((char) i +""+ (char)i); add(tt); }};</pre>
--	---

LAYOUT :- The organisation of a component on a container is known as layout. The awt package has given some class known as layout class which are responsible to organise the components on a container as FlowLayout, BorderLayout, GridLayout, CardLayout.

LayoutManager determines the way components will be arranged inside the container. All container's have their default LayoutManger. However You can apply a different LayoutManager to the containers if required. LayoutManager Classes are Provided in java.awt and javax.swing package.

FlowLayout :- It is the default layout of Applet and Panel. It is the only Layout that provides some spaces between the components. The components added to the container from left to right. When there is no more space available at the right side, then the component will be added from the left to right in the next row. The order in which the components are added will be maintained. If we resize the container then components will be reorganized.

FlowLayout() : it creates an object which will provide some space between different components.

FlowLayout(int orientation) : Same as above but we can provide the orientation & may be LEFT, RIGHT, CENTRE (Default orientation is CENTER).

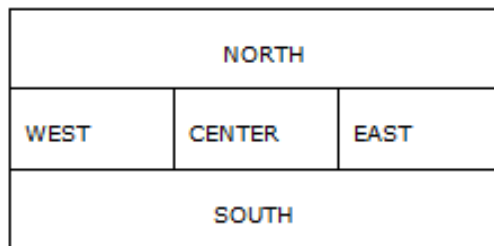
FlowLayout(int orientation, int hgap, int vgap) : hgap is Horizontal gap, vgap is vertical gap.

```
import java.applet.*;
import java.awt.*;
/*<applet code="FlowApp" width="600"
height="500"></applet>*/
public class FlowApp extends Applet {
    public void init() {
        setFont(new Font("asdf", Font.BOLD, 40));
        setBackground( Color.yellow );
        setLayout( new FlowLayout() );
        for(int i =0 ; i <21 ;i++)
            add(new Button( "Ok " + i ));
    }
};
```

```
import java.awt.*;
public class FlowWin extends Frame {
    public FlowWin() {
        setFont(new Font("asdf", Font.BOLD, 40));
        setBackground( Color.yellow );
        setLayout( new FlowLayout() );
        for(int i =0 ; i <21 ;i++)
            add(new Button( "Ok " + i ));
        setSize(300,300);
        setVisible(true);
    }
    public static void main(String args[]) {
        new FlowWin();
    }
}
```

setLayout():- This method is used to remove the layout of a container by the Layout class whose object is given as a parameter to this method.

BorderLayout : - It is the default layout of Frame & Dialog. It divides the container into five regions known as East, West, North, South & Centre as given below



The add() of a container whose Layout is BorderLayout must accept 2 parameter, the 1st parameter must be the region name and the 2nd parameter must be the component. If we do not specify the region name while adding a component, then the component will be added to the center. If we add more than one component to a region then the last component will remain visible. If we resize the container then the component will be resized, but they will never change their position.

If we do not add a component :-

- i) To Central region, then the space of Central region will remain empty.
- ii) To Northern or Southern region, then Center, East, West will occupy these region.
- iii) To Eastern or Western region, then Center will occupy East or West.

BorderLayout() : It creates an object which will not provide any space between components.

BorderLayout(int hgap, int vgap) : It provides horizontal & vertical gap respectively.

```
import java.awt.*;
import java.applet.*;
/*<applet code="BorderApp" width="700"
height="520"></applet>*/
public class BorderApp extends Applet {
    public void init() {
        setLayout( new BorderLayout(15,25) );
        setFont( new Font("Arial", Font.BOLD, 45) );
        setBackground(Color.cyan);
        add( "North", new Button("Button 1") );
        add( "South", new Button("Button 2") );
        add( "East", new Button("Button 3") );
        add( "West", new Button("Button 4") );
        add( "Center", new Button("Button 5") );
        add( "Center", new Button("Button 6") );
    }
};
```

```
import java.awt.*;
public class BorderWin extends Frame {
    public BorderWin() {
        setLayout( new BorderLayout(15,25) );
        setFont( new Font("Arial", Font.BOLD, 45) );
        setBackground(Color.cyan);
        add( "North", new Button("Button 1") );
        add( "South", new Button("Button 2") );
        add( "East", new Button("Button 3") );
        add( "West", new Button("Button 4") );
        add( "Center", new Button("Button 5") );
        add( "Center", new Button("Button 6") );
        setSize(300,300);
        setVisible(true);
    }
    public static void main(String args[]) {
        new BorderWin();
    }
}
```


GridLayout :- It is not the default layout of any container. It will divide a container into number of rows & columns depending upon the rows & column values as specified by the constructor.

The container will be divided like a table, and the addition of components will remain like FlowLayout, but the components will be added to the individual cell of the table. If we resize the container, then the components will also be resized, but they will never change their position.

If the row is non-zero and the column is non-zero in the constructor, then the number of rows will remain fixed where as the no of columns may increase or decrease to accommodate all components, irrespective of column's value.

If the row is zero and the column is non-zero in the constructor, then the no of column will remain fixed where as the no of row may increase to accommodate all components.

If the row and column value is zero in the constructor, then it will generate exception during the runtime.

GridLayout(int rows , int cols) : It will divide a container into rows & columns as specified.

GridLayout(int rows , int cols, int hgap, int vgap)- It is similar to above but provides horizontal & vertical gap.

```
import java.awt.*;
import java.applet.*;
/*<applet code="GridApp" width="700"
height="520"></applet>*/
public class GridApp extends Applet {
    public void init() {
        setLayout(new GridLayout(0,3,25,15));
        setFont(new Font("Afs", Font.BOLD, 45));
        setBackground(Color.cyan);
        for(int i=0; i<18 ; i++)
            add(new Button("Button "+i));
    }
};
```

```
import java.awt.*;
public class GridWin extends Frame {
    public GridWin(){
        setFont(new Font("sadfgas", Font.BOLD, 40));
        setLayout(new GridLayout(0,3,15, 20));
        setBackground(Color.orange);
        for(int i = 0 ; i < 18 ;i++)
            add(new Button("Button " +i));
        setSize(1000,400);
        setVisible(true);
    }
    public static void main(String args[]){
        new GridWin();
    }
};
```

Event :- When the state of a component is changed an event is generated. Each event in java is an object of a predefined Event class. E.g When a Button is clicked an object of ActionEvent class is generated.

Event class & Listener interface :-

Components	Event Class	Listener Interface
Button, MenuItem, List Single Click	Action Event	ActionListener
CheckBox, List, Choice, Radiobutton	ItemEvent	ItemListener
ScrollBar	AdjustmentEvent	AdjustmentListener
TextField	TextEvent	TextListener
Keyboard	KeyEvent	KeyListener
Mouse	MouseEvent	MouseListener, MouseMotionListener
Window	WindowEvent	WindowListener
Focus.	FocusEvent	FocusListener
Container	ContainerEvent	ContainerListener
Component	ComponentEvent	ComponentListener

Methods of Listener interface :-

Listener Interface	Methods (all method of an interface are implicitly public & abstract)
ActionListener	void actionPerformed(ActionEvent ae)
AdjustmentListener	void adjustmentValueChanged(AdjustmentEvent ae)
FocusListener	void focusGained(FocusEvent fe), void focusLost(FocusEvent fe).
ItemListener	void itemStateChanged(ItemEvent ie)
KeyListener	void keyPressed(KeyEvent ke)

	void keyPressed(KeyEvent ke) void keyTyped(KeyEvent ke)
MouseListener	void mouseClicked(MouseEvent me) void mouseEntered(MouseEvent me) void mouseExited(MouseEvent me) void mousePressed(MouseEvent me) void mouseReleased(MouseEvent me)
MouseMotionListener	void mouseDragged(MouseEvent me) void mouseMoved(MouseEvent me)
WindowListener	void windowActivated(WindowEvent we) void windowClosed(WindowEvent we) void windowClosing(WindowEvent we) void windowDeactivated(WindowEvent we) void windowDeiconified(WindowEvent we) void windowIconified(WindowEvent we) void windowOpened(WindowEvent we)
TextListener	void textValueChanged(TextEvent te)
ComponentListener	void componentResized(ComponentEvent ce) void componentMoved(ComponentEvent ce) void componentHidden(ComponentEvent ce) void componentShown(ComponentEvent ce)
ContainerListener	void componentAdded(ContainerEvent ce) void componentRemoved(ContainerEvent ce)

Event Handling:- It is a mechanism which allows us to execute a piece of code at the time of event generation. It is divided into two parts known as Source and Listener.

Source :- It is GUI component that generates an event during the execution due to user's interaction.

Ex- Button, Text field, Mouse etc.

Listener :- It is a predefined interface that allows us to execute a desired task during the event generation.

Ex- for ActionEvent the ActionListener interface is used, similarly for ItemEvent the ItemListener interface is used.

Steps for Event Handling:-

- 1- Import the java.awt.event package because it contains all the predefined class and interfaces required for event handling.

- 2- Create a component to generate the event.

Ex- Button b1 = new Button ("Submit");

- 3- Register the component by following registration () method :-

comp.addZZZ(yyy ob);

Where,

comp is the component which will generate the event.

addZZZ() is the registration method.

ZZZ is the Listener interface name.

ob is an object of a user defined class(yyy) that inherited from the Listener interface to handle the event.

Ex- b1.addActionListener (new Lis());

Choice c1 = new Choice();

c1.addItemListener (new Mis());

If we register a component by the above registration method, then the Listener will be responsible to handle the event. Hence by using the above method we are creating a link or association between the component and the Listener.

If we do not register the component by the above method, then the component will generate the event but there will be no Listener to handle the event.

- 4- Create a user defined class by inheriting from the Listener interface.

```
Ex- class Lis implements ActionListener { }
    class Mis implements ItemListener { }
```

- 5- Override the abstract method of Listener interface in the user defined class and provide the desired task within these overridden method. When exception occurs in try control jumps to catch. Similarly when a component will generate an event, then control jumps to overridden method of the Listener interface.

```
Ex- class Lis implement ActionListener {
    public void actionPerformed (ActionEvent ae) {
        System.out.println("button is clicked");
    }
};
```

e.g. Hadling the Button Event :-

<pre>import java.awt.*; import java.awt.event.*; public class ButEve1 extends Frame { Button b1; TextField tf; public ButEve1() { setFont(new Font("asfas", Font.BOLD, 25)); setBackground(Color.cyan); b1 = new Button("Ok"); tf = new TextField(10); b1.addActionListener(new MyLis()); add("North", tf);</pre>	<pre> add("Center", b1); } class MyLis implements ActionListener { public void actionPerformed(ActionEvent ae){ tf.setText("Button is clicked"); System.out.println("Button is clicked"); } }; public static void main(String rgs[]) { ButEve1 ev = new ButEve1(); ev.setSize(300,100); ev.setVisible(true); } };</pre>
--	---

E.g. Handling Event of Choice & Button

<pre>import java.awt.*; import java.awt.event.*; public class Evt1 extends Frame{ Choice c1, c2; public Evt1() { setFont(new Font("asdf", Font.BOLD, 40)); setBackground(Color.cyan); c1 = new Choice(); c2 = new Choice(); for(int i=1, j=65 ;i<10 ;i++, j++){ c1.add((char)j+""); c2.add(i+""); } add("West", c1); add("East", c2); c1.addItemListener(new It());</pre>	<pre> c2.addItemListener(new It()); setSize(200,200); setVisible(true); } class It implements ItemListener { public void itemStateChanged(ItemEvent ae){ if(ae.getSource()==c1) { int x= c1.getSelectedIndex(); c2.select(x); }else c1.select(c2.getSelectedIndex()); } }; public static void main(String args[]) { new Evt1(); } };</pre>
--	--

getSource() :- This method is available in all predefined event class. It returns the components that has generate the event. We can use it to determine which component has generated the event.

<pre> import java.awt.*; import java.awt.event.*; public class Evt2 extends Frame{ TextField t1, t2; Button b1, b2; Label a1; public Evt2() { setFont(new Font("asdf", Font.BOLD, 40)); setBackground(Color.cyan); b1 = new Button("Submit"); b2 = new Button("Cancel"); t1= new TextField(8); t2= new TextField(8); a1= new Label(" "); setLayout(new GridLayout(0,2,20,15)); add(new Label("User name")); add(t1); add(new Label("Password")); add(t2); add(b1); add(b2); add(a1); b1.addActionListener(new Lis()); b2.addActionListener(new Lis()); } </pre>	<pre> class Lis implements ActionListener { public void actionPerformed (ActionEvent ae){ if(ae.getSource()==b1) { String s1 = t1.getText(); String s2 = t2.getText(); if(s1.equals(s2)) { a1.setForeground(Color.green); a1.setText("success"); }else { a1.setForeground(Color.red); a1.setText("unsuccess"); } }else { t1.setText(""); t2.setText(""); a1.setText(""); } } }; public static void main(String args[]) { Evt2 ev = new Evt2(); ev.setSize(500,300); ev.setVisible(true); }; </pre>
--	--

e.g Handling Mouse Event :-

<pre> import java.awt.*; import java.awt.event.*; public class Mous extends Frame { String str=""; public Mous() { setFont(new Font("Asdf", Font.BOLD, 40)); addMouseMotionListener(new M()); setBackground(Color.cyan); setSize(800,550); setVisible(true); } </pre>	<pre> class M implements MouseMotionListener { public void mouseDragged(MouseEvent me) { } public void mouseMoved(MouseEvent me){ str=me.getX()+" : "+me.getY(); repaint(); }; public void paint(Graphics g) { g.drawString(str, 200,300); } public static void main(String args[]) { new Mous(); } } </pre>
---	--

Event handling by same class :-

<pre> import java.awt.*; import java.awt.event.*; public class Evt3 extends Frame implements ActionListener { Button b1; TextField t1, t2; public Evt3() { setFont(new Font("asdf", Font.BOLD, 40)); b1= new Button("Submit"); t1= new TextField(15); t2= new TextField(15); setLayout(new FlowLayout()); add(t1); add(b1); </pre>	<pre> add(t2); b1.addActionListener(this); /* Here we havegiven 'this' in the registration method, because the same class i.e. Evt3 is inherited from the ActionListener interface to handel the event. */ setSize(400,300); setVisible(true); } public void actionPerformed(ActionEvent ae){ t2.setText("hello " + t1.getText()); } public static void main(String args[]) { Evt3 ev = new Evt3(); } }; </pre>
--	---

Event handling by outer class :-

<pre>import java.awt.*; import java.awt.event.*; public class Evt4 extends Frame { Button b1; TextField t1, t2; public Evt4() { setFont(new Font("asdf", Font.BOLD, 40)); setBackground(Color.cyan); b1= new Button("Submit"); t1= new TextField(15); t2= new TextField(15); setLayout(new FlowLayout()); add(t1); add(b1); add(t2); } }</pre>	<pre>b1.addActionListener(new Lis(this)); /* Here we have given 'this' in the constructor of the Lis class, so that 'this' , i.e. ev object is passed to constructor of Lis class, because Lis class is inherited from the ActionListener interface to handel the event, and Lis class is an Outer class */ setSize(400,250); setVisible(true); } public static void main(String args[]) { Evt4 ev = new Evt4(); } } class Lis implements ActionListener { Evt4 ob; Lis(Evt4 ee){ ob=ee; } public void actionPerformed(ActionEvent ae){ ob.t2.setText("hello " + ob.t1.getText()); } }</pre>
--	--

Event handling by anonymous inner class :-

<pre>import java.awt.*; import java.awt.event.*; public class Evt5 extends Frame { Button b1; TextField t1, t2; public Evt5() { setFont(new Font("asdf", Font.BOLD, 40)); setBackground(Color.cyan); b1= new Button("Submit"); t1= new TextField(15); t2= new TextField(15); setLayout(new FlowLayout()); add(t1); add(b1); add(t2); } /* class Lis implements ActionListener { public void actionPerformed(ActionEvent ae){ t2.setText("hello " + t1.getText()); } }; --eof Nested inner class b1.addActionListener(new Lis()); */</pre>	<pre>/* ActionListener ob= new ActionListener() { public void actionPerformed(ActionEvent ae){ t2.setText("hello " + t1.getText()); } }; --eof Anonymous inner class b1.addActionListener(ob); */ b1.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent ae){ t2.setText("hello " + t1.getText()); } }); setSize(400,250); setVisible(true); } public static void main(String args[]) { Evt5 ev = new Evt5(); } }</pre>
---	--

Adapter Class :- The awt package has given some classes called Adapter class to reduce the burden of a programmer. Adapter classes are concrete classes because they are inherited from the corresponding Listener interface & overridden all the abstract methods. Hence if we inherit from adapter class to handel the event then we may override any number of methods, because the adapter classes are concrete classes, but we are not forced to override all methods. The java.awt.event package has given adapter classes for listener interface consisting of more than one method.

<pre>import java.awt.*; import java.awt.event.*; public class WinEvt extends Frame { public WinEvt() { addWindowListener(new W()); } class W extends WindowAdapter { public void windowClosing(WindowEvent we) {</pre>	<pre>System.out.println("closing..."); System.exit(0); } } public static void main(String args[]) { WinEvt ob = new WinEvt(); ob.setSize(350,175); ob.setVisible(true); } }</pre>
---	---

Audio & Image in Applet :

<pre>import java.awt.*; import java.applet.*; import java.awt.event.*; import java.net.*; /*<applet code="Audio" width=400 height=300></applet>*/ public class Audio extends Applet implements ActionListener { Button b1, b2; URL ob; Image img; AudioClip au; public void init() { setFont(new Font("asfads", Font.BOLD, 40)); ob = getCodeBase(); System.out.println(ob); img = getImage(ob, "abc.gif"); au = getAudioClip(ob, "spacemusic.au"); add(b1= new Button("Play")); add(b2 = new Button("Stop")); } }</pre>	<pre>b1.addActionListener(this); b2.addActionListener(this); } public void paint(Graphics g) { g.drawImage(img, 50,50, 250, img.getHeight(this), this); } public void actionPerformed(ActionEvent ae) { if(ae.getSource() == b1) au.play(); else au.stop(); }};</pre>
--	---

getCodeBase() :- This method of the Applet class is used return the path of the Applet from where it is currently executing as an object of URL class.

URL(Uniform Resource Locator) : This class is available in java.net package. It is used to hold the path of a resource on the internet/intranet. The resource may be a web site or a file on the internet/ intranet.

getAudioClip() :- This method of the Applet class is used to load an audio file with .au/.wav/.midi extension into the memory allotted to the browser. This method will accept the URL of the audio file as the 1st parameter, and the audio file name as a String in the 2nd parameter, so that it will load the audio file and returns a reference of AudioClip interface to access the audio file.

getImage() :- This method of the Applet class is used to load an image file with .gif extension into the memory allotted to the browser. This method will accept the URL of the image file as the 1st parameter, and the image file name as a String in the 2nd parameter, so that it will load the image file and returns an object of Image class to access the image file.

AudioClip interface:- This interface is used to execute a audio file for the Applet. It has some predefined methods as given below :-

play() :- This method will execute the audio file once.

stop() :- This method terminates the execution of audio file.

loop() :- This method will execute the audio file repeatedly.

drawImage() :- This method of the Graphics class is used to draw an image on the Applet. It will accept the object of Image class as the 1st parameter, the 2nd, 3rd, 4th, 5th parameters will accept the left position, right position, width , height of the image respectively. The 6th must be current graphics environment which we have given as 'this'.

Passing Parameters to Applet :-

<pre>/* Store this file with .html extension in the same location where AParam.java is stored */ <html> <body bgcolor="orange"> <h2>Open the source of this file and change the value of param tag, save the file click refresh to see changes.</h2> <applet code="AParama" width="500" height="450"> <param name="t1" value="Parshuram"></pre>	<pre>/* Passing values/parameters from html file to Applet. Store it as AppParam.java */ import java.applet.*; import java.awt.*; public class AppParam extends Applet { String p1,p2="Hello"; public void init(){ p1= getParameter("t1"); p2= getParameter("t2"); int sz = Integer.parseInt(p1); setFont(new Font("fasd",Font.BOLD,sz));</pre>
---	---

<pre><param name="t2" value="78"> </applet> </body></html></pre>	<pre>setBackground(Color.red); } public void paint(Graphics g){ g.drawString(p2, 75,200); };</pre>
--	--

getParameter() :- This method of the Applet class is used to accept any value given in <param> tag to the java program. The param tag is a subtag of applet tag, and the value associated with the param tag will be available to the java program only when we pass the name of the param tag as a parameter to this method. It will the value of param tag as a String object to the java program.

e.g. Applet intercommunication :-

<pre>import java.awt.*; import java.applet.*; import java.awt.event.*; /*<applet code="App1" width=400 height=600 name="App1"></applet> <applet code="App2" width=400 height=600 name="App2"></applet>*/ public class App1 extends Applet implements ActionListener { Button b1; TextArea ta; TextField t1; public void init() { setFont(new Font("asdfads", Font.BOLD, 40)); setLayout(new BorderLayout()); Panel p1= new Panel(); p1.setLayout(new FlowLayout(FlowLayout.LEFT)); p1.setBackground(Color.cyan); p1.add(t1 = new TextField(10)); p1.add(b1 = new Button("send")); add("Center", ta= new TextArea()); add("South", p1); b1.addActionListener(this); } public void actionPerformed(ActionEvent ae) { AppletContext ac = getAppletContext(); App2 a2 = (App2) ac.getApplet("App2"); a2.ta.append(t1.getText()+"\n"); t1.setText(""); } };</pre>	<pre>import java.awt.*; import java.applet.*; import java.awt.event.*; public class App2 extends Applet implements ActionListener { Button b1; TextArea ta; TextField t1; public void init() { setFont(new Font("asdfads", Font.BOLD, 40)); setLayout(new BorderLayout()); Panel p1= new Panel(); p1.setLayout(new FlowLayout(FlowLayout.LEFT)); p1.add(t1 = new TextField(10)); p1.add(b1 = new Button("send")); add("Center", ta= new TextArea()); add("South", p1); b1.addActionListener(this); } public void actionPerformed(ActionEvent ae) { AppletContext ac = getAppletContext(); App1 a1 =(App1)ac.getApplet("App1"); a1.ta.append(t1.getText()+"\n"); t1.setText(""); } };</pre>
--	---

getAppletContext() :- This method of the applet class is used to return the browser's environment to the java program on which the Applet is running, The browser's environment is nothing but the reduced version of JVM available in browser, or it is the run time environment of the appletviewer. To refer the run time environment of the Applet in a java program the AppletContext interface is used.

getApplet() :- This method of the AppletContext interface is used to return an object of the Applet whose name is given as a parameter of this method(It is the same name given in the Applet tag by which the browser will identify the Applet).

CardLayout :- It organizes one component below the other like a deck of card. It will display one component at a time. Hence it manages the space of the container by keeping multiple components in single space.

The add() method of the container whose layout is CardLayout must provide 2 parameter, the 1st parameter must be a String related to the Component, and the 2nd parameter must be the component itself.

The CardLayout class has a method called first() which will display the 1st component added to the container, the next() method will display the next below the current component.

<pre> import java.awt.*; import java.awt.event.*; public class CardDemo extends Frame implements ActionListener { CardLayout cm; Button b[] = new Button[8]; Button bt[] = new Button[4]; Panel p1, p2; public CardDemo() { setFont(new Font("asdf", Font.BOLD, 35)); cm = new CardLayout(); p1 = new Panel(); p2 = new Panel(); p1.setLayout(new GridLayout(2,2)); p2.setLayout(cm); String nam[]={ "First", "Next", "Previous", "Last"}; for(int i=0 ; i<bt.length ; i++) { bt[i] = new Button(nam[i]); bt[i].addActionListener(this); p1.add(bt[i]); } } </pre>	<pre> for(int j=0 ; j<b.length ; j++) { b[j] = new Button("Button: " + j); p2.add(b[j], b[j].getLabel()); } add("West", p1); add("East", p2); setSize(500,300); setVisible(true); } public void actionPerformed(ActionEvent ae){ if(ae.getSource() == bt[0]) cm.first(p2); if(ae.getSource() == bt[1]) cm.next(p2); if(ae.getSource() == bt[2]) cm.previous(p2); if(ae.getSource() == bt[3]) cm.last(p2); } public static void main(String args[]) { new CardDemo(); } } </pre>
--	---

MenuBar class

Constructors:

- public MenuBar()→creates a new menu bar.

Method:

- public Menu add(Menu m)→adds the specified menu to the menu bar.
- public void remove(int index)→removes the menu located at the specified index from this menu bar.
- public void remove(MenuComponent m)→removes the specified menu component from the menu bar.
- public void setHelpMenu(menu m)→sets the help menu on this menu bar to be specified.

Menu class

Constructors:

- public Menu()→creates a menu with an empty label.
- public Menu(String label)→creates a menu with a label as specified

Methods:

- public void addSeparator()→adds a line separator to the menu.
- public MenuItem add(MenuItem mi)→adds the specified menu item to this menu
- public int getItemCount()→gets the no of menu items in this menu
- public MenuItem getItem(int index)→returns the menu item at a specified index.
- public void remove(MenuItem item)→removes a specified menu item from the menu.
- public void removeAll()→removes all the menu items from the menu.

MenuItem class

Constructors:

- public MenuItem()→creates a menu item with no label..
- public MenuItem(String label)→creates a menu item with label.as specified
- public MenuItem(String label, MenuShortcut s)→creates a menu item with specified label and a key board short cut

Methods:

- public void setShortcut(MenuShortcut s)→sets the short cut for the menuitem.
- public MenuShortcut getShortcut()→returns the shortcut menu for the menu item.
- public void setLabel(String label)→sets the label for this menu item.
- public String getLabel()→returns the label of the menu item
- public void setEnabled(boolean b)→sets the menu item to enable if b is true otherwise false.
- public boolean isEnabled()→returns true if the menu item is enabled otherwise false.

CheckboxMenuItem class

Constructors:

- public CheckboxMenuItem()→creates a checkbox menu item with no label.
- public CheckboxMenuItem(String label)→creates a checkbox menu item with specified label.
- public CheckboxMenuItem(String label,boolean state)→creates a checkbox menu item with specified label and specified state.

Methods:

- public void setState(boolean state)→sets the state of the checkbox menu item.
- public boolean getState()→ gets the state of the checkbox menu item.

PopupMenu class

Constructors:

- public PopupMenu()→creates a new popup menu with no label
- public PopupMenu(String label)→creates a new popup menu with specified label

Methods:

- public void show(Component origin, int x,int y)→ shows the popup menu over the origin component at x and y location.
- public void add(Menu m)→adds menu to the popup menu
- public void add(MenuItem mi)→adds menu item to the popup menu
- public void addSeparator()→adds a line separator to the popup menu.

<pre>import java.awt.*; import java.awt.event.*; public class MenuDemo extends Frame{ Label label; MenuBar mb; Menu col,ft,fname,style; MenuItem red,green,blue,font1,font2,font3; MenuShortcut ms; Font font; CheckboxMenuItem style1,style2,style3; public MenuDemo(){ label=new Label("Welcome to Java Menu"); mb = new MenuBar(); col = new Menu("Color"); add(label); ms=new MenuShortcut(KeyEvent.VK_R); red=new MenuItem("Red",ms); green=new MenuItem("Green",new MenuShortcut(KeyEvent.VK_R,true)); blue=new MenuItem("Blue"); col.add(red); col.add(green); col.add(blue); ft = new Menu("Font"); fname= new Menu("Name");</pre>	<pre>font1=new MenuItem("Arial"); font2=new MenuItem("Verdana"); font3=new MenuItem("Garamond"); style= new Menu("Style"); style1= new CheckboxMenuItem("Bold"); style2= new CheckboxMenuItem("Italic"); style3= new CheckboxMenuItem("Underline"); fname.add(font1); fname.add(font2); fname.add(font3); style.add(style1); style.add(style2); style.add(style3); ft.add(fname); ft.add(style); mb.add(col); mb.add(ft); setMenuBar(mb); setSize(300,200); setVisible(true); } public static void main(String args[]){ new MenuDemo(); }}</pre>
--	--

Using Menu & icon :

<pre> import java.awt.*; import java.awt.event.*; public class WinFrm extends Frame { public WinFrm() { super("Win Menu"); setFont(new Font("asfd", Font.BOLD, 40)); MenuItem m1, m2, m3, m4, m5, m6, m7, m8; MenuShortcut ms1= new MenuShortcut(KeyEvent.VK_N, false), ms2= new MenuShortcut(KeyEvent.VK_O, true); m1= new MenuItem("New", ms1); m2= new MenuItem("Open", ms2); m3= new MenuItem("-"); m4= new MenuItem("Save"); m5= new MenuItem("Save as"); m6= new MenuItem("Cut"); m7= new MenuItem("Copy"); m8= new MenuItem("Paste"); Menu mf=new Menu("File"), me= new Menu("Edit"); mf.add(m1); mf.add(m2); mf.add(m3); mf.add(m4); mf.add(m5); me.add(m6); me.add(m7); me.add(m8); MenuBar mb = new MenuBar(); mb.add(mf); mb.add(me); </pre>	<pre> setMenuBar(mb); Panel p1= new Panel(); Panel p2= new Panel(); p1.setLayout(new FlowLayout(FlowLayout.RIGHT)); p2.setLayout(new GridLayout(0,3,15,20)); p1.setBackground(Color.cyan); p2.setBackground(Color.pink); p1.add(new Button("Ok")); p1.add(new Button("Cancel")); for(int i = 0 ; i<100 ; i++) p2.add(new Button("button "+i)); ScrollPane sp = new ScrollPane(); sp.add(p2); add("North",p1); add("Center",sp); add("South", new Button("OK")); add("West", new Button("Cancel")); add("East", new Button("Submit")); Toolkit tk= Toolkit.getDefaultToolkit(); Image img= tk.getImage("abc.gif"); setIconImage(img); setSize(800,550); setVisible(true); } public static void main(String args[]) { new WinFrm(); } </pre>
---	--

Toolkit Class : This object represents the environment on your computer so it encapsulate all the properties and capabilities of that environment , including screen size and resolution. To get Toolkit object we've to use static method as:

```

Toolkit tk = Toolkit.getDefaultToolkit();
OR Frame f= new Frame();
Toolkit tk1=f.getToolkit();

```

GraphicsEnvironment Class: This object contains information about the graphics device attached to the system. We can get the object of GraphicsEnvironment class by :

```
GraphicsEnvironment gp= GraphicsEnvironment.getLocalGraphicsEnvironment();
```

Creating a TextField that accepts only integers :

<pre> import java.awt.*; import java.awt.event.*; public class KeyNum extends Frame { TextField t1; public KeyNum() { setFont(new Font("asfd", Font.BOLD,40)); setBackground(Color.cyan); add("North", new Label("Enter number")); add("Center", t1= new TextField(15)); t1.addKeyListener(new K()); } } </pre>	<pre> class K extends KeyAdapter { public void keyPressed(KeyEvent ke) { if(ke.getKeyCode() >= 57 ke.getKeyCode() <= 48) ke.setKeyCode(KeyEvent.VK_ESCAPE); }; public static void main(String args[]) { KeyNum ke = new KeyNum(); ke.setSize(400,240); ke.setVisible(true); }; } </pre>
---	---

setBounds(int left/x, int top/y, int width, int height) : - This method is used to organize the components according to the pixel value given in the parameter. But before using this method we should remove the layout by providing null to the setLayout().

<pre>import java.awt.*; import java.awt.event.*; public class SetBounds extends Frame { Button b1, b2; TextField t1, t2; Label a1, a2, a3; public SetBounds() { setFont(new Font("asfd", Font.BOLD,30)); setLayout(null); setBackground(Color.cyan); t1= new TextField(9); t2= new TextField(9); t2.setEchoChar('*'); b1= new Button("Submit"); b2= new Button("Cancel"); a1= new Label("User name"); a2= new Label("Password");</pre>	<pre> a3= new Label(" "); a1.setBounds(40,50,170,35); t1.setBounds(220,50,170,35); a2.setBounds(40,100,170,35); t2.setBounds(220,100,170,35); b1.setBounds(100,150,120,35); b2.setBounds(240,150,120,35); a3.setBounds(80,200,400,35); add(a1); add(t1); add(a2); add(t2); add(b1); add(b2); add(a3); setSize(500,270); setVisible(true); } public static void main(String args[]) { new SetBounds(); } }</pre>
--	--

Cursor class

Constructors:

- Cursor (int type) →Creates a new cursor object with the specified type.
- protected Cursor(String name) - Creates a new custom cursor object with the specified name.

Methods:

- public static Cursor getDefaultCursor()→ Returns the system default cursor.
- public int getType()→ Returns the type of the cursor.

Fields: CROSSHAIR_CURSOR DEFAULT_CURSOR E_RESIZE_CURSOR HAND_CURSOR

Example :

```
import java.awt.*;
class CursorDemo extends Frame{
    Button button;
    Cursor cursor;
    CursorDemo(){
        super("Cursor Demo");
        button= new Button("Show Hand");
        setLayout(new FlowLayout());
        cursor= new Cursor(Cursor.HAND_CURSOR);
        button.setCursor(cursor);
        add(button);
        pack();
        setVisible(true);
    }
    public static void main(String args[]){
        new CursorDemo();
    }
}
```