**AWT**

**AWT (Abstract Window Toolkit)**

It is the first approach of java for window programming.

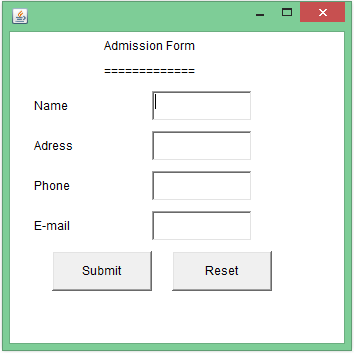
AWT was introduced in the JDK1.1.

**java.awt package**

**java.awt** package and there sub package provides the API (class and interface) for the programming of AWT.

**Two Types of development works are performed in AWT**

1. Designing the GUI (Graphical User Interface) by using classes of **java.awt** package.
2. Making the GUI interface (Event Handling) by using classes and interfaces of **java.awt.event** package.



Label TextFtield

(Normal Component)

Frame (Base /

Container

Component)

Button

**java.lang package**

- String

- System

- Object

- Integer

- Exception

- etc.

**java.awt package**

**-** Component

- Button

- Frame

- TextField

- Label

- etc.

**Component Class is top most class**

In the **java.awt** package there are the numbers of classes known as the component classes.

**There are two types of Component**

1. Container (Base) Component
2. Normal Component

Container component provides the base for the normal components.

Normal components are used to design the GUI.

All the component classes are arranged in the hierarchy.

**java.awt** component class is the top most class for all the components.

**Hierarchy in the java.awt package**

**Java.awt Component**

**Container Button Label TextField Checkbox etc.**

**Window Panel Normal Component**

**Frame Applet**

**Container Component**

**Component class**

Component class provides the basic functionality for all the components.

**Container class**

Container class provides the basic functionality for the containership. It is the abstract class.

**Window**

Window is the child of container class. It provides the facility of the **Title bar** and **border**.

**Frame class**

Frame class is the child of the window and it provides the facility of **menu bar.**

**Panel**

Panel is the hidden container. Panel never help there visibility. In order to make the panel visible we have to add them on to another container.

**Applet**

Applet is the child of the panel. Applets are always gets visible on either on to the web browser on to the applet viewer.

**Methods of component class**

* + - **public void setVisible(boolean b)**

This method is use to make the component visible.

* + - **public void setSize(int weight, int height)**

This method is use to set the size of the component.

* + - **public void setBounds(int x, int y, int weight, int height)**

This method will set the cordinate and the size of the component.

* + - **public void setEnable(boolean b)**

This method is use to enable and disable of the component.

Disable component doesn’t mean that it will not be displayed.

**Methods of the Container class**

* + - **public void add(Component c)**

This method will add the specified the component onto the container.

* + - **public void remove(Component c)**

This method will remove the specified component.

**Steps to create the Frame**

1. Create the object of frame class or its child class.
2. Set the size of the Frame.
3. Makes the frame visible.

**By creating object of Frame**

**//MyFrame.java**

import java.awt.Frame;

class MyFrame

{

Frame fr;

MyFrame( )

{

fr = new Frame( );

fr.setSize(400, 400);

fr.setVisible(true);

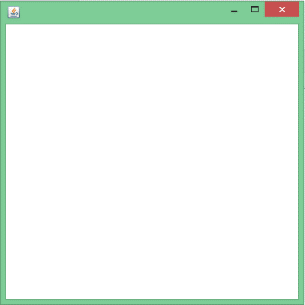
}

public static void main(String s[])

{

new MyFrame( );

}

}

**Output -**

1 new MyFrame(); 2. new Frame( );

fr 110 object of

main( ) Frame

object of MyFrame

**By creating the object of child class of frame**

**//MyFrame.java**

import java.awt.Frame;

class MyFrame1 extends Frame

{

MyFrame1( )

{

setSize(400, 400);

setVisible(true);

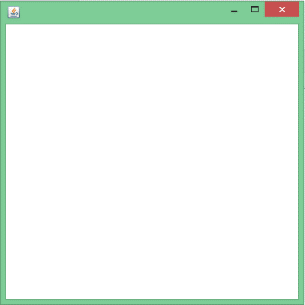
}

public static void main(String s[])

{

new MyFrame1();

}

}

**Output -**

**There are two ways to display the component onto the Frame.**

1. Without Layout Manager
2. With Layout Manager

**Layout Manager**

Layout manager are the predefined arrangements of the components onto the container.

Each and every container such as Frame, Panel, and Applet etc. has the predefined Layout manager default.

**Java.awt** package provides the classes for the layout manager.

Instance of the classes represents the layout manager in the program **java.awt** package.

**- BorderLayout**

**- GridLayout**

**- FlowLayout**

**-CardLayout**

**-etc.**

**Without layout manager** the setBounds() method of the component class is use to set the x and y co-ordinates onto the components.

To remove the predefined layout from the container calling method will be used.

**Eg.**

**fr.setLayout(null);**

**Steps to create and design the Frame without Layout Manager**

1. Create the object of frame or its child class.
2. Remove the predefined Layout.
3. Create the objects of components.
4. Set the bounds (coordinate and size) of the component.
5. Add the component onto the frame.
6. Set the size of the frame.
7. Makes the frame visible.

**//StudentFrame.java**

import java.awt.\*;

class StudentFrame

{

Frame fr;

TextField tf1, tf2;

Label lb1, lb2;

Button b;

StudentFrame( )

{

fr = new Frame( );

fr.setLayout(null);

tf1 = new TextField( );

tf2 = new TextField( );

lb1 = new Label("Name");

lb2 = new Label("Marks");

b = new Button("Submit");

lb1.setBounds(30, 50, 100, 30);

lb2.setBounds(30, 90, 100, 30);

tf1.setBounds(150, 50, 100, 30);

tf2.setBounds(150, 90, 100, 30);

b.setBounds(100, 150, 70, 30);

fr.add(lb1);

fr.add(tf1);

fr.add(lb2);

fr.add(tf2);

fr.add(b);

fr.setSize(300, 200);

fr.setVisible(true);

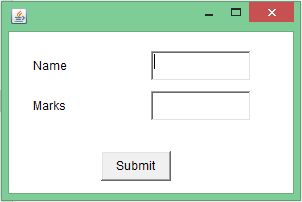
}

public static void main(String s[])

{

new StudentFrame( );

}

}

output-

2 constructor for invoke

Frame

Fr object

lb1 Label

lb2 object

1 new StudentFrame( ); tf1 Label

main ( ) tf2 object

b TextField

object

3 component object

Button TextField

object object

**//Admission.java**

import java.awt.\*;

class Admission

{

Frame fr;

TextField tf1, tf2,tf3,tf4;

Label lb1, lb2,lb3,lb4,lb5,lb6;

Button b1,b2;

Admission()

{

fr = new Frame();

fr.setLayout(null);

tf1 = new TextField();

tf2 = new TextField();

tf3 = new TextField();

tf4 = new TextField();

lb1 = new Label("Admission Form");

lb2 = new Label("=============");

lb3 = new Label("Name");

lb4 = new Label("Adress");

lb5 = new Label("Phone");

lb6 = new Label("E-mail");

b1 = new Button("Submit");

b2 = new Button("Reset");

lb1.setBounds(100,30,200,30);

lb2.setBounds(100,60,200,20);

lb3.setBounds(30,90,100,30);

tf1.setBounds(150,90,100,30);

lb4.setBounds(30,130,100,30);

tf2.setBounds(150,130,100,30);

lb5.setBounds(30,170,100,30);

tf3.setBounds(150,170,100,30);

lb6.setBounds(30,210,100,30);

tf4.setBounds(150,210,100,30);

b1.setBounds(50,250,100,40);

b2.setBounds(170,250,100,40);

fr.add(lb1); fr.add(lb2);

fr.add(lb3); fr.add(tf1);

fr.add(lb4); fr.add(tf2);

fr.add(lb5); fr.add(tf4);

fr.add(lb6); fr.add(tf3);

fr.add(b1); fr.add(b2);

fr.setSize(350,350);

fr.setVisible(true);

}

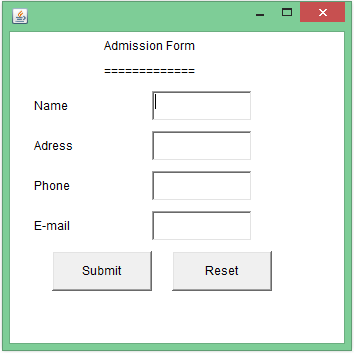
public static void main(String s[ ])

{

new Admission();

}

}

**Output -**

**//Calculator.java**

import java.awt.Button;

import java.awt.Frame;

import java.awt.TextField;

class Calculator

{

Frame fr;

int c=1;

TextField tf;

Button b;

Calculator()

{

fr=new Frame();

fr.setLayout(null);

tf=new TextField();

tf.setBounds(25,50,250,30);

String value[]={"1","2","3","+","4","5","6","-","7","8","9","/",".","0","=","\*"};

Button b[]=new Button[16];

for(int i=0;i<16;i++)

{

b[i]=new Button(value[i]);

fr.add(b[i]);

}

for(int i=0;i<16;i++)

{

fr.add(b[i]);

}

fr.add(tf);

for(int i=0;i<4;i++)

{

c=c+60;

b[i\*4].setBounds(20,(30+c),50,50);

b[i\*4+1].setBounds(90,(30+c),50,50);

b[i\*4+2].setBounds(160,(30+c),50,50);

b[i\*4+3].setBounds(230,(30+c),50,50);

}

fr.setSize(300,350);

fr.setVisible(true);

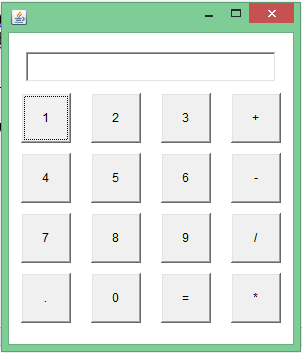
}

public static void main(String []args)

{

new Calculator();

}

}

**Output -**

**Designing of the component through the layout manager**

By default on to the frame **“BorderLayout”** is applicable.

By creating the object of any other layout manager class and passing their reference as the argument in the **setLayout()** method we can change the predefined the layout manager.

**FlowLayout**

This is the default layout manager of the panel and their sub classes.

This layout manager arranges the component from left to right and top to down.

**setBounds()** method will not work in case of any layout manager caption over the components will designed their size.

**setPreferredSized()** method of the component class is use top set the size of the component explicitly.

**Ex: -**

tf1.setPreferredSize(new Dimension(100, 50));

(Reference of TexeField) (class in the **java.awt** package)

**Constructor of the FlowLayout of the class**

* + - **public FlowLayout( )**
    - **public FlowLayout(Int align, int hgap, int vgap)**
    - **public FlowLayout(int align)**

**Vgap –** Vertical gap

**Hgap –** Horizontal gap

Gap will be form in the pixels.

**There can be three type of alignment**

1. center
2. left
3. right

**There are the static constants in the FlowLayout class for the alignment.**

* + - **FlowLayout.CENTER**
    - **FlowLayout.LEFT**
    - **FlowLayout.Right**

**//StudentFrame.java**

import java.awt.\*;

class StudentFrame1

{

Frame fr;

TextField tf1, tf2;

Label lb1, lb2;

Button b;

Panel p;

StudentFrame1( )

{

fr = new Frame( );

fr.setLayout(new FlowLayout( ));

//fr.setLayout(null);

p = new Panel( );

p.setLayout(new FlowLayout( ));

tf1 = new TextField( );

tf2 = new TextField( );

lb1 = new Label("Name");

lb2 = new Label("Marks");

b = new Button("Submit");

//no neet to perform setBounds( );

tf1.setPreferredSize(new Dimension(100, 30));

tf2.setPreferredSize(new Dimension(100,30));

fr.add(lb1);

fr.add(tf1);

fr.add(lb2);

fr.add(tf2);

fr.add(b);

fr.setSize(200, 300);

fr.setVisible(true);

}

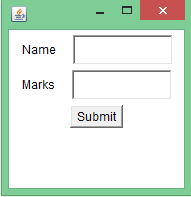
public static void main(String args[])

{

new StudentFrame1( );

}

}



There is the limitation of flowLayout is that as the size of container is change at the runtime the components will be rearranged.

For the flowLayout panel is the appropriate container as there border never be visible.

**//StudentFrame2.java**

import java.awt.\*;

class StudentFrame2

{

Frame fr;

TextField tf1, tf2;

Label lb1, lb2;

Button b;

Panel p;

StudentFrame2( )

{

fr = new Frame( );

fr.setLayout(null);

p = new Panel( );

p.setLayout(new FlowLayout( ));

tf1 = new TextField( );

tf2 = new TextField( );

lb1 = new Label("Name");

lb2 = new Label("Marks");

b = new Button("Submit");

tf1.setPreferredSize(new Dimension(100, 30));

tf2.setPreferredSize(new Dimension(100, 30));

p.add(lb1);

p.add(tf1);

p.add(lb2);

p.add(tf2);

p.add(b);

p.setBounds(30, 50, 200, 300);

fr.add(p);

fr.setSize(400, 400);

fr.setVisible(true);

}

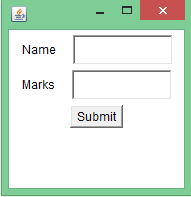
public static void main(String s[])

{

new StudentFrame2( );

}

}



**Event Handling**

* + - Changes in the state of components in the Event.
    - Changes in the state of means changes in the look of components.
    - Changes in the state occur by the user interaction or programmatically.

**Java.awt.Event**

Event is nothing but changes in the state of GUI component is the Event.

In the **java.awt.Event** package there the number of Event class whose objects represents to the Event.

**Event Class**

**- ActionEvent**

**- ItemEvent**

**- FocusEvent**

**- TextEvent**

**- MouseEvent**

**- WindowEvent**

**- etc.**

**ActionEvent**

An instance of this class created when the button is clicked menu item is selected.

**ItemEvent**

An instance of **ItemEvent** is created when the **CheckBox** or **RadioButton** gets clicked or item from the **ComboBox** selected.

**FocusEvent**

By creating the object of this class application gets informed about gaining and lost of the keyboard focus.

**KeyEvent**

An instance of this event class is created to intimate the program about the typing, pressing or releasing the key of the keyboard.

**MouseEvent**

An object of **MouseEvent** class is created to intimate about the clicking, pressing or releasing, dragging etc. of the mouse.

**WindowEvent**

An instance of **WindowEvent** is created when window is open, window is closed, activated, deactivated etc.

**DelegateEvent Model**

In this modelthere are the two participants.

1. Source
2. Listener

**Source**

Source is the component when is responsible for generating the event that means source triggers the event either due to the user interaction or programmatically.

**Listener**

Listener is the object that has something for handling the event.

Listener are represented by the user define classes that has the **callback( )** methods in which the response code for the event handling exist.

**Callback Method**

**Callback** method are the methods whose name provided by the environment and the body of the method prepared by the programmer and then methods are called by the environment.

**main()** is the **callback** method.

**[Note: -** Listener object have to be registered with the source object in order to get the notification.]

As the event generated onto the source object creates the event object and invokes the **callback** method onto the registered listener object.

4 Source Listener Class

object

GUI callback( )

2 7 ------------

-----------

Response

5 Listener 1 Code

3 Object -----------

Event

user object

6

callback( )

1. New object created
2. Listener object gets registered
3. User Interface with the view of source
4. New even object created
5. Source object gets notified
6. Execution of callback method
7. Callback() method invoked and the reference of event object is passed

Corresponding to each event there is one or more interfaces are available known as the listener interfaces.

Developers have to implement the listener corresponding to the event to which developer want to handle.

|  |  |  |
| --- | --- | --- |
| **Event Class** | **Listener Interface** | **Callback Method** |
| **ActionEvent** | ActionListener | public void actionPerformed (ActionEvent e) |
| **ItemEvent** | ItemListener | public void stateChanged(ItemEvent e) |
| **FocusEvent** | FocusListener | public void focusGained(FocusEvent e)  public void focusLost(FocusEvent e) |
| **TextEvent** | TextListener | public void textValueChanged(TextEvent e) |
| **KeyEvent** | KeyListener | public void keyTyped(KeyEvent e)  public void keyPressed(KeyEvent e)  public void keyRealised(KeyEvent e) |
| **MouseEvent** | MouseListener | public void mouseClicked(MouseEvent e)  public void mousePressed(MouseEvent e)  public void mouseRelise(MouseEvent e)  public void mouseEntered(MouseEvent e)  public void mouseExied(MouseEvent e) |
| MouseMotionListener | public void mouseMoved(MouseEvent e)  public void mouseDragged(MouseEvent e) |
| **WindowEvent** | WindowListener | public void windowOpened(WindowEvent e)  public void windowClosed(WindowEvent e)  public void windowClosing(WindowEvent e)  public void windowActivated(WindowEvent e)  public void windowDeactivated(WindowEvent e)  public void windowIconified(WindowEvent e)  public void windowDeiconified(WindowEvent e) |

**//MyListener.java**

import java.awt.event.\*;

public class MyListener implements ActionListener

{

public void actionPerformed(ActionEvent e)

{

System.out.println("Event is generated, now it can be handled");

}

}

Source class provides the registration method to register the listener object.

**Format of the registration method as follows**

* + - **public void add\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_ ref)**

Name of Listener Interface reference variable of listener interface

**Ex:**

Registration method in the button (source) class

* + - **public void addActionListener(ActionListener al)**

Registration method in the Textfield (Source) class

* + - **public void addTextListener(TextListener tl)**

**Working flow of calling the callback method of Listener**

|  |  |
| --- | --- |
| **Java Library** | **User defined** |
| class ActionEvent  {  Component c;  ActionEvent(Component c)  {  this.c = c;  }  public Component getSource( )  {  Return(c);  }  ----------------  }  Class Button  {  ActionListener al;  Public void addActionListener(ActionListener al);  {  This.al=al;  }  <Notification method>  {  **2.3** ActionEvent e=new ActionEvent(this);  //Starting reference id of current source object.  **2.4** Al.actionPerformed(e);  }  } | Class MyListener implements ActionListener  {  Public void actionPerformed(ActionEvent e)  {  **2.5** --------e.getSource  ------------------  }  }  Class MyMain  {  -----------  Mymain( )  {  -------------------------  1.1 Button b=new Button(“ok”);  MyListener ml=new MyListener( );  1.2 b.addActionListener(ml);  }  pubic static void main(String args[])  {  -------------------  }  } |

**2.2**

Ok 2.1 User

* 1. Source object and Listener object created.
  2. Listener object is registered with the source object.
  3. **Button** is clicked by the user.
  4. Notification method is invoked.
  5. Event object is created and the ref id of the current source object is passed.
  6. Callback method invoked and reference of event object is passed.
  7. Execution of callback method started.

**[Note: -** One listener can be registered with the multiple sources.**]**

The event class object is use to identify the source to which the event is generated.

**Method of the Event class**

* + - **public Component getSource()**

This method returns the reference of the source through which this event is generated.

* + - **Public String getActionCommand( )**

This method returns the caption of the source.

**//MyListener.java**

import java.awt.event.\*;

public class MyListener implements ActionListener

{

public void actionPerformed(ActionEvent e)

{

System.out.println("Event is generated, now it can be handled");

}

}

**//EventTest.java**

import java.awt.\*;

import java.awt.event.\*;

class EventTest

{

Frame fr;

Button b1;

EventTest()

{

fr = new Frame();

fr.setLayout(null);

b1 = new Button("First");

MyListener list=new MyListener();

b1.addActionListener(list);

b1.setBounds(60, 80, 100, 30);

fr.add(b1);

fr.setSize(200, 200);

fr.setVisible(true);

}

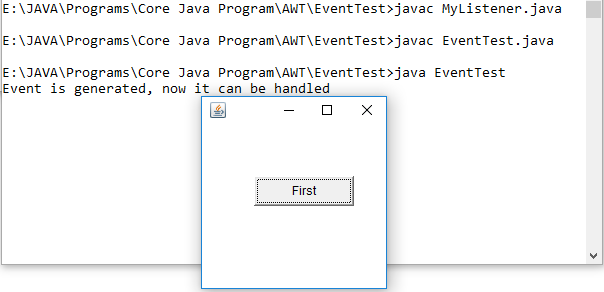
public static void main(String []args)

{

new EventTest();

}

}

****

**Changes in the MyListener class**

**//MyListener.java**

import java.awt.event.\*;

public class MyListener implements ActionListener

{

public void actionPerformed(ActionEvent e)

{

String str = e.getActionCommand();

if(str.equals("First"));

{

System.out.println("First Button is clicked");

}

if(str.equals("Second"));

{

System.out.println("Second Button is clicked");

}

}

}

**//EventTest.java**

import java.awt.\*;

import java.awt.event.\*;

class EventTest

{

Frame fr;

Button b1,b2;

EventTest()

{

fr = new Frame();

fr.setLayout(null);

b1 = new Button("First");

b2 = new Button("Second");

MyListener list=new MyListener();

b1.addActionListener(list);

b2.addActionListener(list);

b1.setBounds(60, 80, 100, 30);

b2.setBounds(180, 80, 100, 30);

fr.add(b1);

fr.add(b2);

fr.setSize(300, 300);

fr.setVisible(true);

}

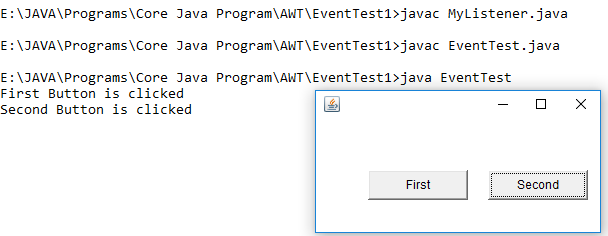
public static void main(String []args)

{

new EventTest();

}

}



**Changes in the EventTest.java**

**//EventTest.java**

import java.awt.\*;

import java.awt.event.\*;

class EventTest

{

Frame fr;

Button b1,b2;

TextField tf;

EventTest()

{

fr = new Frame();

fr.setLayout(null);

b1 = new Button("First");

b2 = new Button("Second");

tf = new TextField();

MyListener listener=new MyListener();

b1.addActionListener(listener);

b2.addActionListener(listener);

b1.setBounds(60, 80, 100, 30);

b2.setBounds(180, 80, 100, 30);

tf.setBounds(60,150,150,30);

fr.add(b1);

fr.add(b2);

fr.add(tf);

fr.setSize(300, 300);

fr.setVisible(true);

}

public static void main(String []args)

{

new EventTest();

}

}

**//MyListener.java**

import java.awt.event.\*;

public class MyListener implements ActionListener

{

public void actionPerformed(ActionEvent e)

{

String str = e.getActionCommand();

EventTest et =new EventTest();

if(str.equals("First"));

{

et.tf.setText("First Button is clicked");

}

if(str.equals("Second"));

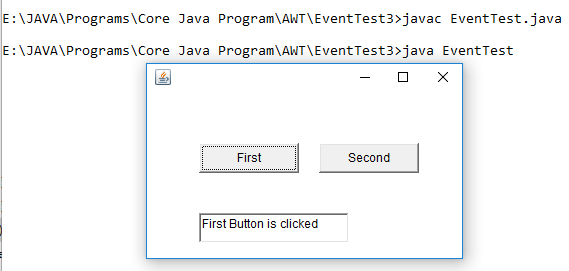
{

et.tf.setText("Second Button is clicked");

}

}

}



**[Note: -** In the event handling there is the always be the common requirement to access the callback methods, so it recommended to make the designer class also as the listener class.**]**

**Changes in the EventTest.java**

**//EventTest.java**

import java.awt.\*;

import java.awt.event.\*;

class EventTest implements ActionListener

{

Frame fr;

Button b1,b2;

TextField tf;

EventTest()

{

fr = new Frame();

fr.setLayout(null);

b1 = new Button("First");

b2 = new Button("Second");

tf = new TextField();

b1.addActionListener(this);

b2.addActionListener(this);

//Registration of the listener with source

b1.setBounds(60, 80, 100, 30);

b2.setBounds(180, 80, 100, 30);

tf.setBounds(60,150,150,30);

fr.add(b1);

fr.add(b2);

fr.add(tf);

fr.setSize(300, 300);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

{

tf.setText("First Button is clicked");

}

if(e.getSource()==b2)

{

tf.setText("Second Button is clicked");

}

}

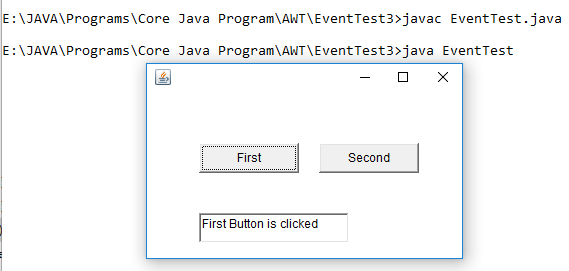
public static void main(String []args)

{

new EventTest();

}

}



Button Object Event Object

110 b1 220

220 Button object has the EventTest object B2 330

As the Listener object

110

330 EventTest has the Button object as the designer component

**Assignment**

**//Calculator.java**

//Calculato.java

import java.awt.\*;

import java.awt.event.\*;

class Calculator //implements ActionListener

{

Frame fr;

Label num1,num2,result;

TextField txtn1,txtn2,txtr;

Button add,sub,mul,div;

Calculator()

{

fr=new Frame();

fr.setLayout(null);

txtn1=new TextField( );

txtn2=new TextField( );

txtr=new TextField( );

num1=new Label("First Number");

num2=new Label("Second Number");

result=new Label("Result");

add=new Button("Add");

sub=new Button("Sub");

mul=new Button("Mul");

div=new Button("Div");

add.addActionListener(this);

sub.addActionListener(this);

mul.addActionListener(this);

div.addActionListener(this);

num1.setBounds(20,30,100,30);

txtn1.setBounds(130,30,100,30);

num2.setBounds(20,60,100,30);

txtn2.setBounds(130,60,100,30);

result.setBounds(20,140,100,30);

txtr.setBounds(130,140,100,30);

add.setBounds(20,100,50,30);

sub.setBounds(80,100,50,30);

mul.setBounds(140,100,50,30);

div.setBounds(200,100,50,30);

fr.add(num1); fr.add(num2); fr.add(result);

fr.add(txtn1); fr.add(txtn2); fr.add(txtr);

fr.add(add); fr.add(sub); fr.add(mul); fr.add(div);

fr.setSize(300,250);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

int x,y,res;

x=Integer.parseInt(txtn1.getText());

y=Integer.parseInt(txtn2.getText());

if(e.getSource()==add)

{

res=x+y;

txtr.setText(""+res);

}

if(e.getSource()==sub)

{

res=x-y;

txtr.setText(""+res);

}

if(e.getSource()==mul)

{

res=x\*y;

txtr.setText(""+res);

}

if(e.getSource()==div)

{

res=x/y;

txtr.setText(""+res);

}

}

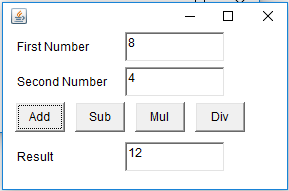
public static void main(String []args)

{

new Calculator();

}

}



**Implementing the window listener**

**//WindowEventTest.java**

import java.awt.\*;

import java.awt.event.\*;

class WindowEventTest implements WindowListener

{

Frame fr;

TextField tf;

WindowEventTest()

{

fr=new Frame();

fr.setLayout(null);

tf=new TextField();

tf.setBounds(30,50,200,30);

fr.add(tf);

fr.setSize(300,300);

fr.setVisible(true);

fr.addWindowListener(this);

}

public void windowOpened(WindowEvent e)

{

System.out.println("Window Open");

}

public void windowClosed(WindowEvent e)

{

System.out.println("Window is closed");

fr.dispose();

}

public void windowClosing(WindowEvent e)

{

System.out.println("Window close");

}

public void windowActivated(WindowEvent e)

{

System.out.println("Window is activated");

}

public void windowDeactivated(WindowEvent e)

{

System.out.println("Window is deactivated");

}

public void windowIconified(WindowEvent e)

{

System.out.println("Window is minimized");

}

public void windowDeiconified(WindowEvent e)

{

System.out.println("Window is maximized");

}

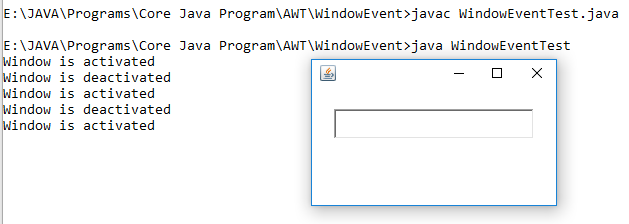
public static void main(String []args)

{

new WindowEventTest();

}

}



**Adaptor classes**

In the **java.awt.event** package there are the adaptor classes corresponding to the entire listener interface that have more than one callback methods adaptor class implements to the listener interface and provides the default implementation to all the callback methods.

In the adaptor classes there are the body of all the callback body we can prepare the listener class also by extending the adaptor classes.

**Ex:**

* + - **WindowAdaptor**
    - **MouseAdaptor**
    - **MouseMotionAdaptor**
    - **KeyAdaptor**
    - **FocusAdaptor**
    - **Etc.**

**WindowListener**

**class MyListener**

(Direct implementation of listener Interface)

**windowListener**

**class WindowAdaptor**

**class MyListener**

(Indirect implementation of Listener)

**SWING**

It is extension of AWT, it was introduced in JDK 1.2 as the part of java foundation classes (JFC).

JFC means three things.

**JFC = AWT + Swing + java 2D**

**Differences between the AWT and Swing (Advantages of the Swing over AWT)**

1. Swing component are light weighted component with aspect of memory and execution time while AWT components are the heavy weight components.
2. At runtime AWT components take their definition from native Operating System (C & C++) while the Swing component purely developed in java.
3. Look and feel of AWT component very across the platform while the look and feel of swing component remains same across Operating System.
4. Awing provides advance components while the AWT provides basic components.
5. Swing utilized the MVC architecture while the AWT is just only the toolkit.

MVC 🡪

**java.swing Package**

Provides API of Swing Hierarchy of Swing components

Java.awt.Compnent

Java.awt.Container

Window Panel JComponent

Frame Applet

JTextComponent

JFrame JApplet IinternalFraame JTextField

JTextArea

JPasswordField

Heavy weight JButton

Component JLabel

JComboBox

Light weight JTable

Component JList

JMenu

JMenuBar

Etc.

**AWT stands for abstract?**

In the AWT library there the abstract class named toolkit at the runtime there implemented class created by the JRE according to the Operating System that implemented class will be the proxy class.

Contain the code of JNI to interact

With the window GUI library

class Linux

class Window JNI certain native code to interact

Toolkit with Linux GUI

Contains function is Linux GUI Toolkit

native language library

Let suppose window GUI library has the method with the name getColoy(), setColor() and Linux GUI library hasthe method setBackground().

Abstract class Toolkit

Window GUI library abstract void setBackground();

void setColor() class WindowToolkit class WindowTookit

{ { {

------------- void setBackcolor() void setBackcolor()

} { {

Invoking the invoking the

setColor() using JNI setBackground() using JNI

} }

} }

Linux GUI library

void setBackground()

{

-------

}

**[Note: -** Corresponding to each component of the AWT there is the extra class known as the peer class the use of peer class is to interact with GUI library of the Operating System through the toolkit class.**]**

**Button b=new Button(“ok”);**

**1** Button object

Object of

Ok **2** peer Button object peer Button

Created

Toolkit object intercepted

Invoking function to provide the information producing

Of Button such as setColor caption function **5**

Native GUI name toolkit

**7** Library object

Ok view is prepared getting the function name

**MVC (Model View and Controller)**

This is the design according to that the whole functionality is divided into the three loosely coupled layer.

1. Model
2. View
3. Controller

**Model**

Contains the data use to prepare the view.

**View**

It is responsible together the view by using the data provided by the model.

**Controller**

It is responsible to interact with the view and model layer that means the controller is responsible to coordinate between the model views.

**JButton b=new JButton(“Ok”);**

1. 2.4

Object of model

JButton

2.6 1.3 2.3 1.4 2.7

1.1

Controller user

2.2 2.1

1.2 2.5

View 1.5 Ok

2.8

1.0 JButton object is created.

1.1 Model view and controller object are created.

1.2 View just notified to generate the view.

1.3 Model is requested to provide the data.

1.4 Data is provided to the view layer.

1.5 View is generated.

2.1 Button is clicked.

2.2 Controller is intercepted.

2.3 Model is notified to change the data.

2.4 Data is changed by the model.

2.5 View is notified to update the view.

2.6 Model is requested for new data.

2.7 Updated data is provided.

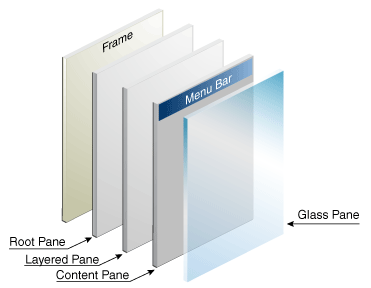
2.8 View gets updated.

**javax.swing.JFrame**

**JFrame**

It is container available in the swing API all the containers in the swing are heavy weighted that means they take their definition from the native Operating System at the runtime.

To retain the layered architecture on the swing is available.



**Root Pane**

Provides the base that will interact with the native GUI library

**Layered Pane**

Provides the depth

**Content Pane**

All the component always be added on to the content pane.

**Glass Pane**

Provides the facility of the drag and drop

**[N0ote: -** Up to the JDK 1.4 we can’t directly add the components on to the JFrame rather we have to first of all get the content pane then we have to add the components onto the content pane.**]**

**Up to the JDK 1.4**

**Container c=fr.getContainer();**

**c.add(b);**

To add any Button

**JDK 1.5**

**Fr.add(b);**

To add Button indirectly onto the Content Pane

**Method of JFrame**

* + - **public void setDefaultCloseOperation(int closingMade)**

Unlike the Frame of the AWT the case button of the JFrame can worked in the different ways by specifying the integer value in the argument we can set the closing modes.

There are the static constants.

**Swing constants interface for the close mode.**

**SwingConstants.EXIT\_ON\_CLOSE**

**SwingConstants.DO\_NOTHING\_ON\_CLOSE**

**SwingConstants.HIDE\_ON\_CLOSE**

**SwingConstants.DISPOSE\_ON\_CLOSE**

Variables by default

**Ex:**

int x=5;

fr.setDefaultCloseOperation(SwingConstents.EXIT\_ON\_CLOSE);

interface SwingConstants

{

public static final int EXIT-ON\_CLOSE = -;

public static final int HIDE\_ON\_CLOSE = -;

JFrame class implements to the SwingConstants interface that’s why we can use the constants also with the JFrame class.

**fr.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**java.swing.JLabel**

**JLabel**

This is the component through which any text or image can displayed onto the GUI.

**Constructor**

* + - **public JLabel(String text)**
    - **public JLabel(String text, int alignment)**

**StaticConstants for the alignment**

**JLabel.CENTER**

**JLabel.LEFT**

**JLabel.RIGHT**

* + - **public JLabel(Icon i)**

Through this constant the image can be placed onto the JLabel.

**Icon**

It is the abstract class ImageIcon is the can child class of Icon.

**To set the Image on the Labal**

new JLabel(new ImageIcon(“abc.jpg”));

//relative path

new JLabel(new ImageIcon(“d:\\temp\\abc.jpg”));

//abstract path

* + - **public JLabel(String text, Icon I, int alignment)**

**Method of JLabel**

* + - **public void setText(String str)**

It will set the text in the label

* + - **public String getText()**

This method will return the text of the JLabel

* + - **public void setItem(Icon i)**

This method is use to set the icon onto the JLabel.

* + - **public Icon getIcon()**

It will returns the reference of the Icon of the JLabel

* + - **public int getHorizontalAlignment()**

This method returns the horizontal alignment of JLabel which is currently set.

* + - **public void setHorizontalAlignment(int alignment)**

This method will set the horizontal alignment.

**//JLabelTest.java**

import javax.swing.\*;

import java.awt.event.\*;

public class JLabelTest implements ActionListener

{

JFrame fr;

JLabel lb1;

JButton b1, b2, b3;

JLabelTest()

{

fr = new JFrame();

fr.setLayout(null);

lb1 = new JLabel();

b1 = new JButton("First");

b2 = new JButton("Second");

b3 = new JButton("Third");

lb1.setBounds(20,20,300,300);

b1.setBounds(20,330,100,30);

b2.setBounds(130,330,100,30);

b3.setBounds(240,330,100,30);

b1.addActionListener(this);

b2.addActionListener(this);

b3.addActionListener(this);

fr.add(lb1); fr.add(b1);

fr.add(b2); fr.add(b3);

fr.setSize(400,450);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

{

lb1.setIcon(new ImageIcon("Apple(2).jpg"));

}

if(e.getSource()==b2)

{

lb1.setIcon(new ImageIcon("Apple(2)ja.jpg"));

}

if(e.getSource()==b3)

{

lb1.setIcon(new ImageIcon("Apple-Logo.jpg"));

}

}

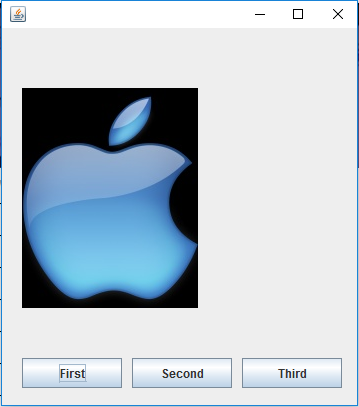
public static void main(String []args)

{

new JLabelTest();

}

}



**JTextField**

This component use to take the input from the end user this is to accept only the single line of text.

**Constructor**

* + - **public JTextField()**
    - **public JTextField(String str)**
    - **public JTextField(int noOfColumns)**

This constructor only be use in case of layout manager.

**Methods of JTextField**

* + - **public void setText(String str)**
    - **public String getText()**
    - **public String getSelectedText()**
    - **public void setectAll()**

It is use to select all the TextField.

* + - **public void selset(int startIndex, int endIndex)**

This method will select the text from the specified start index to the specific end index.

* + - **public void copy()**

This method will copy the selected text and placed onto the clipboard.

* + - **public void paste()**

This method will paste the text into this text firld at the current position of the cursor from the clipboard.

* + - **public void cut()**

This method will cut the selected text and placed in the keyboard.

* + - **public void setFont(Font f)**

This method is use to set the font.

* + - **public void setdBackground(Color c)**

It is use to set background color

* + - **public void setForground(Color c)**

It is use to set forground color.

* + - **public void setSelectedTextColor(Color c)**

* + - **public void setSelectionColor(Color c)**

**java.awt.Font class**

As instance of this is use to set the font

**Constructor**

* + - **public Font(String style, int font, int size)**

**Ex:-**

tf.setFont(new Font(“Arial Balck”, Font.BOLD, 25));

**Font.BOLD**

**Font.ITALIC**

**Font.PLAIN**

**java.awt.Color class**

This class instance is use to represent the calor.

**Constructor**

* + - **public Color(int r, int g, int b)**

**value 0 – 255 , 0 – 255 , 0 - 255**

**Ex: - new Color(0,0,255); //pure blue**

**[Note: -** In the color class there are the color static color objects are exits.**]**

class Color

{

public static final Color RED = new Color(255,0,0);

public static final Color red = new Color(255,0,0);

public static final Color GREEN = new Color(0,255,0);

}

**Some more color static object**

**Color.YELLOW**

**Color.GRAY**

**Color.PINK**

**Color.ORANGE**

**Color.etc**

**Ex: tf.setBackground(new Color(255,0,0)); or**

**tf.setBackgroun(Color.RED);**

**JTextArea**

It is also the important component, through this component multiple lines of text can be entered.

Constructor and methods are same as the JTextField.

**Additional Constructor**

* + - **public JTextArea(int noOfRows, int noOfColumns)**

This constructor always be used in case of layout manager.

**Additional Methods**

* + - **public void append(String str)**

This method is use to append the specified text in text of this TextArea.

* + - **public void setLineWrap(boolean b)**

If argument value is true then the automatically horizontal scrollbar will be not append.

**[Note: -** In the swing there is no facility of scrollbar by default rather we have to add the scrollbar explicitly.**]**

* + - **public void replaceSelection(String str)**

This method will replace the selected text with the specified text.

**javax.swing.JScrollPane class**

There is no scrollbar on any component in swing, we have to add it explicitly.

It is the container like component it has the horizontal and Vertical scrollbars implicitly.

If we want to apply the scrollbars on any component then we have to add that component onto the ScrollPane.

**Constructor**

* + - **public JScrollPane(Component c)**
    - **public JScrollPane(Component c, int vsbPolicy, int hsbpolicy)**

**Static constants for policy**

**JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED**

**JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS**

**JScrollPane.VERTICAL\_SCROLLBAR\_NEVER**

**JScrollPane.HORIXONTAL\_SCROLLBAR\_AS\_NEED**

**JScrollPane.HORIXONTAL\_SCROLLBAR\_ALWAYS**

**JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER**

**//TestArea.java**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class TextAreaTest implements ActionListener

{

JFrame fr;

JTextArea ta;

JButton b;

JScrollPane pane;

TextAreaTest()

{

fr=new JFrame();

fr.setLayout(null);

ta=new JTextArea(); //step 1

pane=new JScrollPane(ta,JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS); //step 2

pane.setBounds(30,30,200,200);

b=new JButton("Click");

b.setBounds(30,270,100,50);

fr.add(pane);

fr.add(b);

b.addActionListener(this);

fr.setSize(300,300);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

ta.requestFocusInWindow();

ta.select(3,5);

}

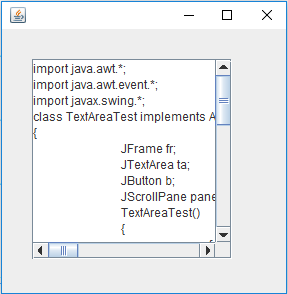
public static void main(String s[])

{

new TextAreaTest();

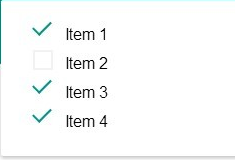
}

}



**JCheckBox**

This also the input type component but it always takes the value in the form of Boolean value either true or false.



**Constructor**

* + - **public JCheckBox(String str)**

It will make the checkbox with specified text

**Ex:**

**new JCheckBox("Java");**



**public JCheckBox(String str, Icon i)**

It will add Image

**Method of JCheckBox**

* + - **public void setSelected(boolean b)**

This method is use to make the checkbox selected.

* + - **public void getText()**
    - **public void setSelectedIcon(Icon i)**

This will set the icon that will display when the checkbox is selected.

**Note:** On the checkbox the ActionEvent and ItemEvent both are generated.

**//JCheckBoxTest.java**

import java.awt.event.\*;

import javax.swing.\*;

class JCheckBoxTest implements ActionListener

{

JFrame fr1,fr2;

JCheckBox cb1,cb2,cb3,cb4;

JLabel lb;

JButton b1,b2;

JCheckBoxTest()

{

fr1=new JFrame();

fr2=new JFrame();

fr1.setLayout(null);

fr2.setLayout(null);

cb1=new JCheckBox("Mobile");

cb2=new JCheckBox("Laptop");

cb3=new JCheckBox("IPod");

cb4=new JCheckBox("All");

lb=new JLabel();

b1=new JButton("Get Bill");

b2=new JButton("Exit");

cb1.setBounds(30,30,100,30);

cb2.setBounds(30,60,100,30);

cb3.setBounds(30,90,100,30);

cb4.setBounds(30,120,100,30);

b1.setBounds(140,150,100,30);

lb.setBounds(50,50,150,30);

b2.setBounds(70,120,100,30);

fr1.add(cb1);fr1.add(cb2);fr1.add(cb3);fr1.add(cb4); fr1.add(b1);

fr2.add(lb) ; fr2.add(b2);

cb4.addActionListener(this);

b1.addActionListener(this);

b2.addActionListener(this);

fr1.setSize(300,300);

fr2.setSize(250,250);

fr1.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==cb4)

{

if(cb4.isSelected())

{

cb1.setSelected(true);

cb2.setSelected(true);

cb3.setSelected(true);

}

else

{

cb1.setSelected(false);

cb2.setSelected(false);

cb3.setSelected(false);

}

}

if(e.getSource()==b1)

{

int amt=0;

if(cb1.isSelected())

amt+=5000;

if(cb2.isSelected())

amt+=20000;

if(cb3.isSelected())

amt+=1000;

lb.setText("Payable Amount= "+amt);

fr2.setVisible(true);

}

if(e.getSource()==b2)

System.exit(0);

}

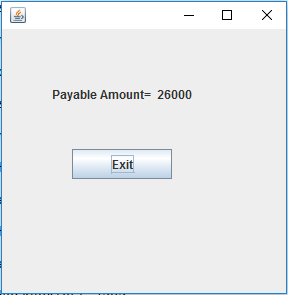
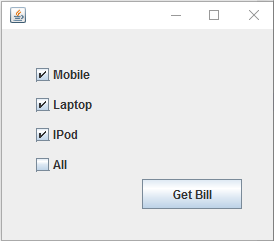
public static void main(String s[])

{

new JCheckBoxTest();

}

}



**JRadioButton**

This is also the input component uses to take the input from the end user in the form of Boolean value.

JRadioButton are same as JCheckBox but it can be grouped.

Constructor and method are same as JCheckBox

**ButtonGroup class**

This class is use to group the radio buttons.

**Ex:**

**JRadioButton rb1,rb2,rb3;**

**------------- creating the objects of RadioButton ---------------**

**ButtonGroup bg=new ButtonGroup();**

**bg.add(rb1);**

**bg.add(rb2);**

**bg.add(rb3);**

**r1.setBounds(-,-,-,-);**

**r2.setBounds(-,-,-,-);**

**r3.setBounds(-,-,-,-);**

**fr.add(rb1);**

**fr.add(rb2);**

**fr.add(rb3);**

**JComboBox**

This is also the input component that provides of list of values in front of end-user.

**Constructor**

* + - **public JComboBox()**
    - **public JComboBox(Object []values)**

Ex: String city[]={"Noida","Agra","Kanpur","Delhi"};

JComboBox cb=new JComboBox(city);

* + - **public JComboBox(Vector v)**

This constructor will take the collection object that contains the values for the ComboBox.

* + - **public JComboBox(ComboBoxModel model)**

ComboBoxModel is the interface and its implemented class is the DefaultComboBoxModel.

An instance of the ComboBoxModel represent to the values for the ComboBox.

**Ex: -**

DefaultComboBoxModel model=new DefaultComboBoxModel();

model.addElement("Noide"):

model.addElement("Delhi");

model.addElement("Agra");

JComboBox cb=new JComboBox(model);

**Method of JComboBox**

* + - **public Object getSelectItem()**

This method will return the object selected on the comboBox.

* + - **public int getSelectedIndex()**

This method will return the index of selected item.

* + - **public void setSelectedIndex(int index)**

This method will select the specified item.

* + - **public void setSelectedItem(Object item)**

* + - **public void setModel(ComboBoxModel model)**

This method will set the model.

* + - **public void setVisibleRowCount(int numberOfRows)**

This method will set the number of rows in the ComboBox that will be visible when the ComboBox gets open.

* + - **public void addItem(Object ob)**

This method will add the specified item in the ComboBox.

* + - **public void removeItem(Object ob)**

This method will remove the specified item from the comboBox.

**Methods of DefaultComboBoxModel**

* + - **public void addElement(Object element)**
    - **public void removeElementAt(int index)**
    - **public void removeAllElement()**
    - **public int getSize()**

This method returns the number of element contended by the ComboBox.

**//ComboBoxTest.java**

import javax.swing.\*;

import java.awt.event.\*;

class ComboBoxTest implements ActionListener

{

JFrame fr;

JComboBox cb1,cb2;

DefaultComboBoxModel model1,model2,model3;

ComboBoxTest()

{

fr=new JFrame();

fr.setLayout(null);

String city[]={"Noida","Agra","Kanpur"};

model1=new DefaultComboBoxModel(city);

model2=new DefaultComboBoxModel();

model2.addElement("Bhopal");

model2.addElement("Jabalpur");

model2.addElement("Indore");

model3=new DefaultComboBoxModel();

model3.addElement("Jaipur");

model3.addElement("Ajmer");

model3.addElement("Udaypur");

String state[]={"UP" , "MP" , "Raj"};

cb1=new JComboBox(state);

cb2=new JComboBox();

cb1.setBounds(50,50,100,30);

cb2.setBounds(180,50,100,30);

fr.add(cb1);fr.add(cb2);

cb1.addActionListener(this);

fr.setSize(300,300);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==cb1)

{

String st=(String)cb1.getSelectedItem();

if(st.equals("UP"))

cb2.setModel(model1);

if(st.equals("MP"))

cb2.setModel(model2);

if(st.equals("Raj"))

cb2.setModel(model3);

}

}

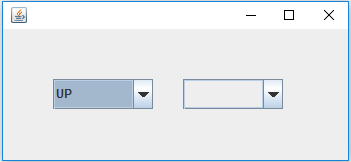
public static void main(String s[])

{

new ComboBoxTest();

}

}



**JList**

**java.swing.JList class**

JList is also input type component like JComboBox. In the JList multiple item can be viewed on to the GUI.

At a time user can select single item or the multiple items by pressing ctrl key.

Constructor and Methods are same as the JComboBox.

**Additional Methods**

* + - **public void setSelectionMode(int mode)**

This method will set the selection mode of the element on the JList.

**ListSelectionModel Interface**

ListSelectionModel interface provides the static constants for selection mode.

**ListSelectionModel.SINGLE\_SELECTION**

**ListSelectionModel.SINGLE\_INTERVAL\_SELECTION**

**ListSelectionModel.MULTIPLE\_INTERVAL\_SELECTION**

**Ex: -**

jl.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

* + - **public Object getSelectedValue()**
    - **public Object[] getSelectedValues()**
    - **public int getSelectedIndex()**
    - **public int[] getSelectedIndices()**

**javax.swing.event.ListSelectionEvent class**

This event is generated when the any list item is selected.

**Listener Interface: ListSelectionListener**

**Callback Method**

* + - **public void valueChanged(ListSelectionListener e)**

**//ListTest.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.event.\*;

class ListTest extends MouseAdapter implements ListSelectionListener , ActionListener

{

JFrame fr1,fr2;

JList list1,list2;

JScrollPane pane1,pane2;

DefaultListModel model1,model2;

JButton badd,bget;

JLabel lb1,lb2,lb3;

int price[]={1500,800,4500,5000};

String values[];

ListTest()

{

fr1=new JFrame();

fr2=new JFrame();

fr1.setLayout(new FlowLayout());

fr2.setLayout(new FlowLayout());

values=new String[]{"Keyboard","Mouse","HD","Monitor"};

model1=new DefaultListModel();

for(int i=0 ; i<values.length ; i++)

model1.addElement(values[i]);

list1=new JList(model1);

pane1=new JScrollPane(list1);

model2=new DefaultListModel();

list2 = new JList(model2);

pane2=new JScrollPane(list2);

pane1.setPreferredSize(new Dimension(100,250));

pane2.setPreferredSize(new Dimension(100,250));

badd=new JButton("Add");

bget=new JButton("Get Bill");

lb1=new JLabel("double click to remove");

lb2=new JLabel();

lb3=new JLabel();

lb2.setPreferredSize(new Dimension(300,50));

lb3.setPreferredSize(new Dimension(300,50));

fr1.add(pane1);

fr1.add(badd);

fr1.add(pane2);

fr1.add(lb1);

fr1.add(lb2);

fr1.add(bget);

badd.addActionListener(this);

bget.addActionListener(this);

list1.addListSelectionListener(this);

list2.addMouseListener(this);

fr2.add(lb3);

fr1.setSize(400,400);

fr1.setVisible(true);

fr2.setSize(300,300);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==badd)

{

Object values[]=list1.getSelectedValues();

for(int i=0 ; i<values.length ;i++ )

{

model2.addElement(values[i]);

}

}

if(e.getSource()==bget)

{

int i=0,bill=0;

while(true)

{

try

{

String str=(String)model2.getElementAt(i);

int index=model1.indexOf(str);

bill=bill + price[index];

i++;

}

catch(ArrayIndexOutOfBoundsException ee)

{

break;

}

}

lb3.setText("Total Paybale amount: "+bill);

fr2.setVisible(true);

}

}

public void valueChanged(ListSelectionEvent e)

{

int i=list1.getSelectedIndex();

lb2.setText("Price of Selected Item: "+price[i]);

}

public void mouseClicked(MouseEvent e)

{

if(e.getSource()==list2)

{

System.out.println("mouse clicked");

if(e.getClickCount()==2)

{

model2.removeElement(list2.getSelectedValue());

}

}

}

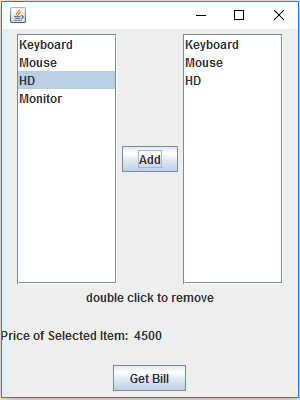
public static void main(String s[])

{

new ListTest();

}

}



**JTable**

JTable is use to show the report on to the GUI in the tabular format that means in form of row and columns.

**Constructor**

* + - **public JTable(int noOfRows, int noOfColumns)**
    - **public JTable(TableModel model)**
    - **public JTable(String [][]values, String []heading)**
    - **eg.**

**String [][]values={{“ABC”,”101”,”A”},{“XYZ”,”102”,B}};**

**String []heading={“Name”,”Rollno”,”Grade”};**

**JTable jt = new JTable(values , heading);**

**Note: TableModel Interface exist in the javax.swing.table package.**

**Methods**

* + - **public Object getValueAt(int rowNo, int colNo)**
    - **public void setValueAt(String value, int rowNo, int colNo)**
    - **public void setModel(TableModel model)**

**[Note: -** Table heading will only be visible when the JTable will be added onto the JScrollPane otherwise we have to create the separate object of JTable header.**]**

**//TabelTest.java**

import javax.swing.\*;

import javax.swing.table.\*;

import java.awt.\*;

import java.awt.event.\*;

class TableTest implements ActionListener

{

JTable table;

JScrollPane pane;

JTextField tf1,tf2,tf3,tf4;

JButton b1,b2;

JLabel lb1,lb2,lb3,lb4;

DefaultTableModel model;

JFrame fr1,fr2;

TableTest()

{

fr1=new JFrame();

fr2=new JFrame();

fr1.setLayout(new FlowLayout());

fr2.setLayout(new FlowLayout());

lb1=new JLabel("Name");

lb2=new JLabel("Rollno");

lb3=new JLabel("Marks");

lb4=new JLabel("Grade");

tf1=new JTextField(10);

tf2=new JTextField(10);

tf3=new JTextField(10);

tf4=new JTextField(10);

b1=new JButton("Add");

b2=new JButton("Show");

fr1.add(lb1); fr1.add(tf1);

fr1.add(lb2); fr1.add(tf2);

fr1.add(lb3); fr1.add(tf3);

fr1.add(lb4); fr1.add(tf4);

fr1.add(b1); fr1.add(b2);

model=new DefaultTableModel();

model.addColumn("Name");

model.addColumn("Rollno");

model.addColumn("Marks");

model.addColumn("Grade");

table=new JTable(model);

pane = new JScrollPane(table);

pane.setPreferredSize(new Dimension(200,250));

fr2.add(pane);

fr1.setSize(300,300);

fr2.setSize(300,300);

fr1.setVisible(true);

b1.addActionListener(this);

b2.addActionListener(this);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

{

String str[]=new String[4];

str[0]=tf1.getText();

str[1]=tf2.getText();

str[2]=tf3.getText();

str[3]=tf4.getText();

model.addRow(str);

tf1.setText("");tf2.setText("");

tf3.setText("");tf4.setText("");

}

if(e.getSource()==b2)

fr2.setVisible(true);

}

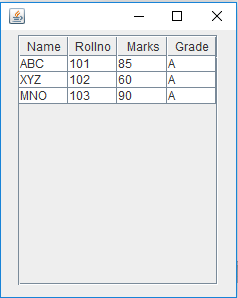
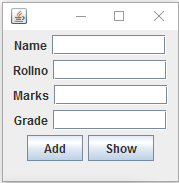
public static void main(String s[])

{

new TableTest();

}

}



**DialogBoxes**

DialogBoxes are the intermediate window that is use to show the intermediate result in front of end user.

**There are three types of DialogBoxes**

1. Builtin DialogBox
2. Custom DialogBox
3. File DialogBox

**Builtin DialogBox**

Builtin DialogBox are also three types

1. Message DailogBox
2. Confirmation DialogBox
3. Input DialogBox

**javax.swing.JOptionPane class**

javax.swing.JOptionPane class provides the static method to show the different type of Builtin DialogBox

* + - **public static void showMessageDialog(Component parent, String title, String message, int msgType)**

**Static constants for the message type**

**JOptionPane.INFORMATION\_MESSAGE**

**JOptionPane.QUESTION\_MESSAGE**

**JOptionPane.WARNNING\_MESSAGE**

The first argument is the parent from on which this DialogBox will be appeared.

* + - **Public static int showConfiemDialog(Component parent, Sting title, String message, int optionType)**

**Static constant for the Option types**

**JOptionPane.YES\_NO\_OPTION**

**JOptionPane.YES\_NO\_CANCLE\_OPTION**

**JOptionPane. OK\_CANCLE\_OPTION**

**Static constant for the Option types that will be return by this method**

**JOptionPane.YES\_OPTION**

**JOptionPane.NO\_OPTION**

**JOptionPane. OK\_OPTION**

**JOptionPane. CANCLE\_OPTION**

This method will show the confirmation dialogBox.

The return value of this method denotes the option selected by the user.

* + - **public static String showInputDialog(Component parent, String defaultMessage, String message)**

This method will show the inputDialogBox.

If user clicked on the ok Button then this method returns the text entered by the user.

If cancel button is clicked then this method returns the null.

**//DialogBoxTest.java**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

class DialogBoxTest implements ActionListener

{

JFrame fr;

JButton b1,b2,b3;

JTextField tf;

DialogBoxTest()

{

fr=new JFrame();

fr.setLayout(new FlowLayout());

b1=new JButton("Message");

b2=new JButton("Confirm");

b3=new JButton("Input");

tf=new JTextField(15);

fr.add(tf);

fr.add(b1);

fr.add(b2);

fr.add(b3);

b1.addActionListener(this);

b2.addActionListener(this);

b3.addActionListener(this);

fr.setSize(200,250);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

{

String name=tf.getText();

JOptionPane.showMessageDialog(fr,"Hello & welcome "+name ,"Sample Dialog Box", JOptionPane.WARNING\_MESSAGE);

}

if(e.getSource()==b2)

{

int x=JOptionPane.showConfirmDialog(fr,"Java Question","Swing is better than AWT" , JOptionPane.YES\_NO\_CANCEL\_OPTION);

if(x==JOptionPane.YES\_OPTION)

tf.setText("Correct answer");

if(x==JOptionPane.NO\_OPTION)

tf.setText("Incorrect answer");

if(x==JOptionPane.CANCEL\_OPTION)

tf.setText("you should know it");

}

if(e.getSource()==b3)

{

String name=JOptionPane.showInputDialog(fr,"Enter Name","Enter your name");

if(name!=null)

tf.setText("Welcome "+name);

else

tf.setText("Welcome unknown");

}

}

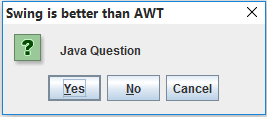
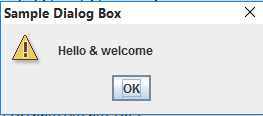
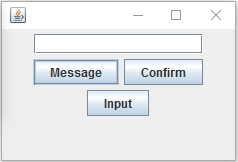
public static void main(String s[])

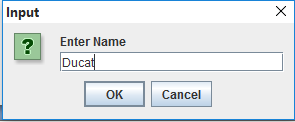
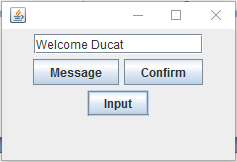
{

new DialogBoxTest();

}

}





**CustomDialogBox**

Through the custom DialogBox are used to unused to custom DialogBox.

**javax.swing.JDialog class**

javax.swing.JDialog class is used to prepare the custom DialogBox.

**[Note: -** DialogBoxes never have minimized and maximize button their Titlebar.**]**

**//CustomDialogBox.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

class CustomDialogTest implements ActionListener

{

JFrame fr;

JDialog d;

JButton b;

JTextField tf;

CustomDialogTest()

{

fr=new JFrame();

fr.setLayout(new FlowLayout());

d=new JDialog(fr,"My Dialogbox",true);

d.setLayout(new FlowLayout());

tf=new JTextField("This is the Custom dialog box");

b=new JButton("Show");

b.addActionListener(this);

fr.add(b);

d.add(tf);

d.setSize(200,200);

fr.setSize(300,300);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

d.setVisible(true);

}

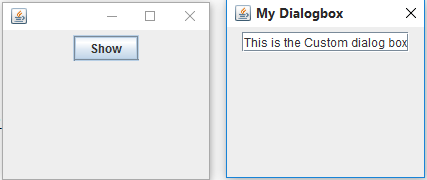
public static void main(String s[])

{

new CustomDialogTest();

}

}



**FileDialogBox**

JFileChooser class in the javax.swing package provides the method to show the FileDialogBoxes.

**There are two Types of FileDialogBoxes**

1. Open DialogBox
2. Save DialogBox

**Method**

* + - **public int showOpenDialog(Frame parent)**
    - **public int showSaveDialog(Frame parent)**

**Working of the FileChooser**

Java Object of

Application 1.1 FileChooser 1.3

MyDocument

1.2 ----------------

----------------

----------------

2.3 File

* 1. File Open Cancel

2.2 2.1

User

1.1 New Object created

1.2 showOpenDialog();

1.3 DialogBox opened

2.1 File Selected

2.2 Open Button clicked

2.3 File object created

2.4 File content retrieved

**//FileInputOutputTest.java**

import java.io.\*;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

class FileInputOutputTest implements ActionListener

{

JFrame fr;

JTextArea ta;

JScrollPane pane;

JButton b1,b2;

FileInputOutputTest()

{

fr=new JFrame();

fr.setLayout(new FlowLayout());

ta=new JTextArea(20,30);

pane=new JScrollPane(ta);

b1=new JButton("Open");

b2=new JButton("Save");

b1.addActionListener(this);

b2.addActionListener(this);

fr.add(pane);

fr.add(b1);

fr.add(b2);

fr.setSize(300,300);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

{

try

{

JFileChooser fch=new JFileChooser();

int x=fch.showOpenDialog(fr);

if(x==JFileChooser.APPROVE\_OPTION)

{

File f=fch.getSelectedFile();

BufferedReader br=new BufferedReader(new FileReader(f)); while(true)

{

String str=br.readLine();

if(str==null)

break;

ta.append(str+"\n");

}

}

}

catch(Exception ee)

{

ee.printStackTrace();

}

}

if(e.getSource()==b2)

{

try

{

JFileChooser fch=new JFileChooser();

int x=fch.showSaveDialog(fr);

if(x==JFileChooser.APPROVE\_OPTION)

{

File f=fch.getSelectedFile();

String str=ta.getText();

PrintWriter pw=new PrintWriter(f);

pw.println(str);

pw.flush();

}

}

catch(Exception ee)

{

ee.printStackTrace();

}

}

}

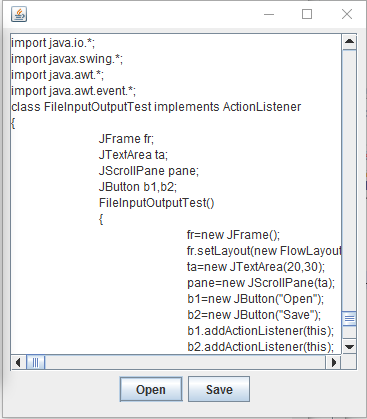
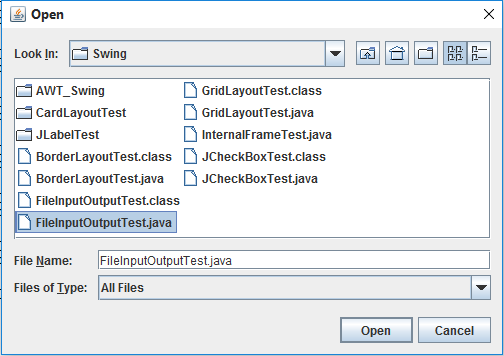
public static void main(String s[])

{

new FileInputOutputTest();

}

}



**LayoutManagers**

layoutManage are the predefined arrangements of the component on to the container.

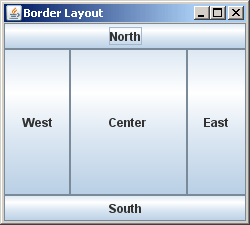
**Some layoutManagers are**

* + - **FlowLayout**
    - **BorderLayout**
    - **GridLayout**
    - **CardLayout**

**BorderLayout**

BorderLayout is the default layout of the frame and its subclasses.

BorderLayout arrange the components in the five different regions north, south, west and center.



In one region only one component can be placed.

There is the overloaded to add the component in specific region.

* + - **public void add(Component c, int region)**

**Static constants for the region**

* + - **BorderLayout.NORTH**
    - **BorderLayout.SOUTH**
    - **BorderLayout.EAST**
    - **BorderLayout.WEST**
    - **BorderLayout.CENTER**

**Constructors of BorderLayout**

* + - **public BorderLayout();**
    - **public BorderLayout(int horizontalSpace, int verticalSpace)**

**//BorderLayoutTest.java**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class BorderLayoutTest implements ActionListener

{

JFrame fr;

JButton b1,b2,b3,b4;

JTextField tf;

BorderLayoutTest()

{

fr=new JFrame();

fr.setLayout(new BorderLayout());

b1=new JButton("North");

b2=new JButton("South");

b3=new JButton("East");

b4=new JButton("West");

tf=new JTextField();

fr.add(b1,BorderLayout.NORTH);

fr.add(b2,BorderLayout.SOUTH);

fr.add(b3,BorderLayout.EAST);

fr.add(b4,BorderLayout.WEST);

fr.add(tf,BorderLayout.CENTER);

b1.addActionListener(this);

b2.addActionListener(this);

b3.addActionListener(this);

b4.addActionListener(this);

fr.setSize(300,300);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

tf.setText("North selected");

if(e.getSource()==b2)

tf.setText("South selected");

if(e.getSource()==b3)

tf.setText("East selected");

if(e.getSource()==b4)

tf.setText("West selected");

}

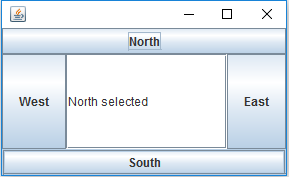
public static void main(String s[])

{

new BorderLayoutTest();

}

}



**GridLayout**

GridLayout arrange the components in the form of grids one grid can contain only component.

Each grid always has equal size.

Grids automatically field with the left to right and with the top to bottom manner.

**GridLayout**

**Constructor**

* + - **public GridLayout()**
    - **public GridLayout(int noOfRows, int noOfColumns)**

In case of 1st constructor only the single Grid with one row and one column will be created.

**//GridLayoutTest.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

class GridLayoutTest implements ActionListener

{

JFrame fr;

JPanel p1,p2;

JButton b[]=new JButton[16];

JButton b1,b2;

int count=0;

GridLayoutTest()

{

fr=new JFrame();

p1=new JPanel();

p2=new JPanel();

p1.setLayout(new GridLayout(4,4));

fr.setLayout(new FlowLayout());

for(int i=0 ; i<16 ; i++)

{

b[i]=new JButton(""+(i+1));

b[i].setVisible(false);

p1.add(b[i]);

}

b1=new JButton("Get");

b2=new JButton("Reset");

b1.addActionListener(this);

b2.addActionListener(this);

p2.add(b1);

p2.add(b2);

p1.setPreferredSize(new Dimension(300,100));

p2.setPreferredSize(new Dimension(100,100));

fr.add(p1);

fr.add(p2);

fr.setSize(400,400);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

{

while(count<16)

{

int x=(int)(Math.random()\*16);

System.out.println(x);

if(b[x].isVisible())

continue;

b[x].setVisible(true);

count++;

break;

}

}

if(e.getSource()==b2)

{

for(int i=0 ; i<16 ; i++)

b[i].setVisible(false);

count=0;

}

}

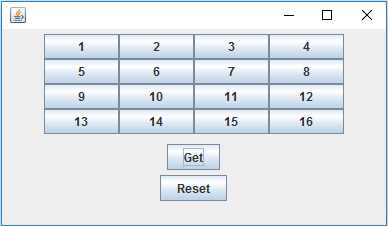
public static void main(String s[])

{

new GridLayoutTest();

}

}



**CardLayout**

It arranges the component in the form of back of cards.

Each card can contains the only one component and at the onetime only one card can be visible.

The card layout can’t be applicable onto the frame or their subclasses card layout applicable on to the pane.

Jpane

cards

When ever we add the new component onto the panel that has the CardLayout then automatically the new card will be created and the new component will be added onto it.

**Ex:** Reference of Panel

**p.add(b);**

**1.1** new

Card created Reference of theButton

View of Panel

Ok 1.3 cards added onto

The Panel

There is the overloaded add method to add the component onto the contains that hasthe CardLayout.

**Signature**

* + - **public void add(Component c, String cardName)**

**Methods of CardLayout**

* + - **public void show(String cardName,Component parent)**

This will show the specified card.

* + - **public void first(Component parent)**

This method will show the first card.

* + - **public void last(component c)**

This method show last card

* + - **public void next()**
    - **public void previous()**

**//CardLayoutTest.java**

import java.awt.event.\*;

import javax.swing.\*;

import java.awt.\*;

class CardLayoutTest implements ActionListener

{

JFrame fr;

JButton b1,b2,b3,b4;

JLabel lb1,lb2,lb3,lb4;

JPanel p1,p2;

CardLayout cd;

CardLayoutTest()

{

cd=new CardLayout();

fr=new JFrame();

fr.setLayout(new FlowLayout());

p1=new JPanel();

p2=new JPanel();

p2.setLayout(cd);

b1=new JButton("First");

b2=new JButton("Prev");

b3=new JButton("Next");

b4=new JButton("Last");

lb1=new JLabel(new ImageIcon("1.jpg"));

lb2=new JLabel(new ImageIcon("2.jpg"));

lb3=new JLabel(new ImageIcon("3.jpg"));

lb4=new JLabel(new ImageIcon("4.jpg"));

p1.add(b1);p1.add(b2);p1.add(b3);p1.add(b4);

p2.add(lb1,"first");

p2.add(lb2,"second");

p2.add(lb3,"third");

p2.add(lb4,"fourth");

b1.addActionListener(this);

b2.addActionListener(this);

b3.addActionListener(this);

b4.addActionListener(this);

fr.add(p1);

fr.add(p2);

fr.setSize(1024,800);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

{

cd.first(p2);

}

if(e.getSource()==b2)

cd.previous(p2);

if(e.getSource()==b3)

cd.next(p2);

if(e.getSource()==b4)

cd.last(p2);

}

public static void main(String s[])

{

new CardLayoutTest();

}

}

**Working with the Menus**

In the javax.swing package there are the three classes for the Menus

1. JMenuBar
2. JMenu
3. JMenuItem

Menu



MenuBar

MenuItem

**JMenuBar**

This class represents the member onto the frame.

**Constructor**

* + - **public JMenuBar()**

**[Note: -** setJMenuBar() of the JFrame class is use to set the JMenuBar onto the JFrame.**]**

**Method of JMenuBar**

* + - **public void add(JMenu m)**

It will add the JMenu onto the MenuBar.

**JMenu class**

An instance of JMenu class is used to represent the menu onto the Frame.

* + - **public JMenu(String title)**

**Method of JMenu**

* + - **public void add(JMenu item)**

This will add the MeunItem on to the menu

* + - **public void setMnemonic(int key)**

This method is use to set the shortcut key with the combination of “Alt” key.

There are the static constants for the key in the keyEvent class.

**KeyEvent class**

**KeyEvent.VK\_A**

**KeyEvent.VK\_B**

**KeyEvent.VK\_C**

**KeyEvent.VK\_D**

**Etc**

**JMenuItem**

An instance of this class is represents the MenuItem that will added onto the JMenu.

**Constrctor**

* + - **public JMenuItem()**
    - **public JMenuItem(String title)**
    - **public JMenuItem(Icon i)**

**[Note: -** add() method of the JMenu is use to add the MenuIten onto the menu.**]**

**Method of JMenuItem**

* + - **public void setAcceleration(KeyStroke k)**

This method is use to set the shortcut on the MenuItem.

KeyStroke class has the static factory method to create and provide their object.

* + - **public static KeyStroke getKeyStroke(int key, int modifier)**

**Static constants for key**

**KeyEvent.VK\_A**

**KeyEvent.VK\_B**

**Etc**

**Static constant for the modifier**

**KeyEvent.SHIFT\_MASK**

**KeyEvent.CTRL\_MASK**

**KeyEvent.ALT\_MASK**

**Ex: -** KeyStroke ks = KeyStroke.getKeyStroke(KeyEvent.VK\_A, KeyEvent.SHIFT\_MASK);

mi.setAcceleration(ks);

**[Note: -** On the MenuItem actionEvent is generated as the MenuItem is clicked.**]**

**Steps to Applied the Menu on JVM**

1. Create the object to MenuBar
2. Create the object of JMenu
3. Create the object of JMenuItem
4. Add the MenuItem on the JMenu
5. Add the Menu on to the MenuBar
6. Set the MenuBar on to the JFrame

**//MenuTest.java**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

class MenuTest implements ActionListener

{

JFrame fr;

JMenuBar bar;

JMenu m;

JMenuItem mi1,mi2,mi3;

JLabel lb;

MenuTest()

{

fr=new JFrame();

fr.setLayout(null);

bar=new JMenuBar();

m=new JMenu("Color");

mi1=new JMenuItem("Red");

mi2=new JMenuItem("Green");

mi3=new JMenuItem("Blue");

mi1.addActionListener(this);

mi2.addActionListener(this);

mi3.addActionListener(this);

m.add(mi1);m.add(mi2);m.add(mi3);

bar.add(m);

fr.setJMenuBar(bar);

lb=new JLabel("Select the color to change the color of the frame");

lb.setBounds(10,50,300,50);

fr.add(lb);

m.setMnemonic(KeyEvent.VK\_C);

mi1.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_R,KeyEvent.SHIFT\_MASK));

fr.setSize(400,400);

fr.setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

Container cr=fr.getContentPane();

if(e.getSource()==mi1)

cr.setBackground(Color.red);

if(e.getSource()==mi2)

cr.setBackground(Color.green);

if(e.getSource()==mi3)

cr.setBackground(Color.blue);

}

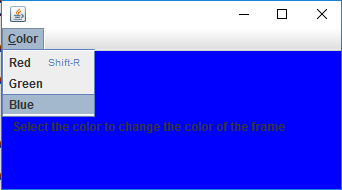
public static void main(String s[])

{

new MenuTest();

}

}



**m.addSeparator();**

m.addSeparator is used to add the separator in the menu in between the menu items.

**JPopupMenu**

JPopupMenu are the menu that will be displayed upon the any component as the result of any KeyEvent.

**//JPopupMenu**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

class PopupMenuTest extends MouseAdapter implements ActionListener

{

JFrame fr;

JPopupMenu menu;

JMenuItem m1,m2,m3;

JTextField f1;

PopupMenuTest()

{

fr=new JFrame();

fr.setLayout(null);

f1.setBounds(40,50,50,80);

m1=new JMenuItem("Red");

m2=new JMenuItem("Green");

m3=new JMenuItem("Blue");

menu=new JPopupMenu();

menu.add(m1);menu.add(m2);menu.add(m3);

m1.addActionListener(this);

m2.addActionListener(this);

m3.addActionListener(this);

fr.addMouseListener(this);

fr.setSize(400,400);

fr.setVisible(true);

}

public void mouseClicked(MouseEvent e)

{

if(e.getButton()==3)

{

Point p=e.getPoint();

int x=(int)p.getX();

int y=(int)p.getY();

menu.show(fr , x, y); // to show the popup menu

}

}

public void actionPerformed(ActionEvent e)

{

Container c=fr.getContentPane();

if(e.getSource()==m1)

c.setBackground(Color.red);

if(e.getSource()==m2)

c.setBackground(Color.green);

if(e.getSource()==m3)

c.setBackground(Color.blue);

}

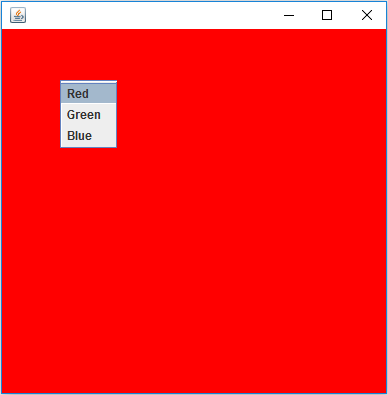
public static void main(String s[])

{

new PopupMenuTest();

}

}



**JSlider class**

This is the component that is use to get the input from the end-user by sliding.



**Constructor**

* + - **public JSlider()**

It will make the slide object and with the one and 100 as the minimum and maximum value with the default slider position at the 50.

* + - **public JSlider(int max, int min, int value)**

This constructor is use to JSlider object with the specified minimum, maximum and default value.

**Method**

* + - **public int getValue()**

This method returns the current value of the slider.

* + - **public void setValue(int val)**

This method is use to set the value of the slider.

* + - **public void setPaintTicks(boolean b)**

This method if the argumented value is true then Ticks will be displayed on the slider but before the calling of this method the calling method must be invoked.

* + - **public void setMinorTicksSpacing(int x)**
    - **public void setPaintLabels(boolean b)**
    - **public void setSnapToTicks(boolean b)**

**[Note: -** On the JSlider as the value gets Changed immediately javax.swing.event.ChangeEvent is occured.**]**

**Listener is ChangeListener (I)**

**Callback Method**

* + - **public void stateChanged(ChangeEvent e)**

**//SliderTest.java**

import javax.swing.\*;

import javax.swing.event.\*;

import java.awt.\*;

class SliderTest implements ChangeListener

{

JFrame fr;

JSlider slider;

JLabel lb;

JRadioButton rb1,rb2,rb3;

SliderTest()

{

fr=new JFrame();

fr.setLayout(new FlowLayout());

slider=new JSlider(0,255);

slider.setMajorTickSpacing(20);

slider.setPaintTicks(true);

slider.setPaintLabels(true);

slider.setPreferredSize(new Dimension(500,50));

lb=new JLabel("Use the slider to change the color");

rb1=new JRadioButton("Red");

rb2=new JRadioButton("Green");

rb3=new JRadioButton("Blue");

ButtonGroup bg=new ButtonGroup();

bg.add(rb1); bg.add(rb2);

bg.add(rb3);

slider.addChangeListener(this);

fr.add(lb); fr.add(slider);

fr.add(rb1);fr.add(rb2);fr.add(rb3);

fr.setSize(600,300);

fr.setVisible(true);

}

public void stateChanged(ChangeEvent e)

{

int x=slider.getValue();

Container c=fr.getContentPane();

if(rb1.isSelected())

c.setBackground(new Color(x,0,0));

if(rb2.isSelected())

c.setBackground(new Color(0,x,0));

if(rb3.isSelected())

c.setBackground(new Color(0,0,x));

}

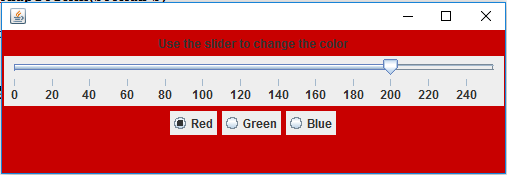
public static void main(String s[])

{

new SliderTest();

}

}



**javax.swing.Timer class**

An instance of this class is used to set the timer in the swing application.

Timer object always be registered with the ActionListener and after the specified time automatically invokes the ActionEvent.

**Constructor**

* + - **public Timer(int Miliseconds, ActionListener al)**

**Method of the Timer**

* + - **public void start()**

This method will start to generate the ActionEvent

* + - **public void stop()**

This method will stop generation of the ActionEvent.

**//TimerTest.java**

import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

class TimerTest implements ActionListener

{

JFrame fr;

JButton b1,b2;

JLabel lb1,lb2;

Timer t;

Panel p1,p2;

TimerTest()

{

fr=new JFrame();

fr.setLayout(new FlowLayout());

b1=new JButton("Start timer");

b2=new JButton("Stop timer");

lb1=new JLabel("Hello");

lb2=new JLabel("Welcome");

t=new Timer(500 , this);

p1=new Panel();

fr.add(b1); fr.add(b2);

fr.add(lb1); fr.add(lb2);

Toolkit t=Toolkit.getDefaultToolkit();

Dimension d=t.getScreenSize();

int a=(int)d.getWidth();

int b=(int)d.getHeight();

fr.setSize(a,b);

fr.setVisible(true);

b1.addActionListener(this);

b2.addActionListener(this);

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==t)

{

String temp=lb1.getText();

lb1.setText(lb2.getText());

lb2.setText(temp);

}

if(e.getSource()==b1)

t.start();

if(e.getSource()==b2)

t.stop();

}

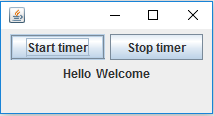
public static void main(String s[])

{

new TimerTest();

}

}



**JColorChooser**

This is component that provides the color pane from that we can select the color.

An instance of JColorChooser automatically always associated with the colorSelectionModel instance and we have to register the change listener with that model.

As the color gets selected on the colorChooser an immediately change event is generated and by the following method of the colorSelectionModel we can get the selected color.

**//ColorTest.java**

import javax.swing.\*;

import javax.swing.event.\*;

import javax.swing.colorchooser.\*;

import java.awt.\*;

class ColorChooserTest implements ChangeListener

{

JFrame fr;

JColorChooser ch;

ColorSelectionModel model;

ColorChooserTest()

{

fr=new JFrame();

fr.setLayout(null);

ch=new JColorChooser(); // step 1

model = ch.getSelectionModel(); // step 2

model.addChangeListener(this);// step 3

ch.setBounds(50,50,400,400);

fr.add(ch);

fr.setSize(500,500);

fr.setVisible(true);

}

public void stateChanged(ChangeEvent e)

{

Color c=model.getSelectedColor();

fr.getContentPane().setBackground(c);

}

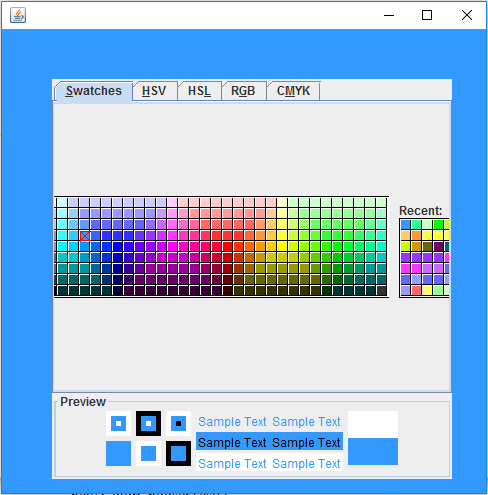
public static void main(String s[])

{

new ColorChooserTest();

}

}



**Signature**

* + - **public Color getSelectedColor()**

**JInternalFrame**

JInternalFrame is the container that is always be open within the frame.

**Steps to prepare the InternalFrame**

1. Create to prepare the InternalFrame
2. Create the object of DesktopPane
3. Set the desktopPane as the contentPane on the Frame
4. Add the InternalFrame on to the DesktopPane.

**Constructor of JInternalFrame**

* + - **public JInternalFrame(String title)**
    - **public JInternalFrame(String title, boolean closeable, boolean resizable, boolean meximizable, boolean minimizable)**

**//InternalFrameTest.java**

import javax.swing.\*;

class InternalFrameTest

{

JFrame fr;

JInternalFrame ifr;

InternalFrameTest()

{

fr=new JFrame();

fr.setLayout(null);

//step 1

ifr=new JInternalFrame("Test" ,true , true ,true ,true);

ifr.setBounds(50,50,200,200);

ifr.setVisible(true);

// step 2

JDesktopPane pane=new JDesktopPane();

//step 3

fr.setContentPane(pane);

//step 4

pane.add(ifr);

fr.setSize(300,300);

fr.setVisible(true);

}

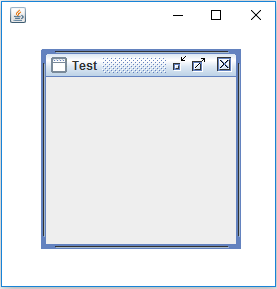
public static void main(String s[])

{

new InternalFrameTest();

}

}



**JTree**

JTree is the component that is used to display the content in the tree structure format.

DefaultMutableTreeNode class in the Javax.swing package interface of this class works as the node of the tree.

**Constructor**

* + - **public JTree(DefaultMultableTreeNode Root)**

**//JTreeTest.java**

import javax.swing.\*;

import javax.swing.tree.\*;

import java.awt.\*;

class TreeTest

{

JFrame fr;

JScrollPane pane;

JTree tree;

TreeTest()

{

fr=new JFrame();

fr.setLayout(null);

DefaultMutableTreeNode root=new DefaultMutableTreeNode("Root");

DefaultMutableTreeNode color=new DefaultMutableTreeNode("Color");

DefaultMutableTreeNode red=new DefaultMutableTreeNode("Red");

DefaultMutableTreeNode green=new DefaultMutableTreeNode("Green" );

DefaultMutableTreeNode blue=new DefaultMutableTreeNode("Blue");

DefaultMutableTreeNode yellow=new DefaultMutableTreeNode("yellow");

tree=new JTree(root);

pane=new JScrollPane(tree);

color.add(red);

color.add(green);

color.add(blue);

color.add(yellow);

root.add(color);

//tree.setRoot(root);

pane.setBounds(30,50,150,250);

fr.add(pane);

Toolkit t=Toolkit.getDefaultToolkit();

Dimension d=t.getScreenSize();

int width=(int)d.getWidth();

int height=(int)d.getHeight();

fr.setSize(width,height);

fr.setVisible(true);

}

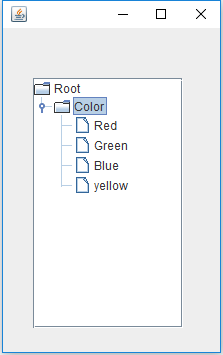
public static void main(String s[])

{

new TreeTest();

}

}



**JProgressBar**

This is the component which is used to show the progress of the any task.

**Constructor**

* + - **public JProgressBar(int min, int max)**
    - **public JProgressBar()**

By using the default constructor the progressbar will have their default value one and 100 as the min value and max value.

**Method**

* + - **public void setValue(int value)**

Up to the specified value the progressbar will be field.

* + - **public void setString(String str)**

By using this method the argument string will be displayed on to the progress bar.

**//ProgressBar.java**

import javax.swing.\*;

import java.awt.\*;

class ProgressTest

{

JFrame fr1,fr2;

JProgressBar pbar;

ProgressTest()

{

fr1=new JFrame();

fr1.setLayout(new FlowLayout());

fr2=new JFrame();

pbar=new JProgressBar();

pbar.setPreferredSize(new Dimension(150,50));

pbar.setStringPainted(true);

fr1.add(pbar);

fr1.setSize(250,150);

fr1.setVisible(true);

fr2.setSize(300,300);

for(int i=1 ; i<=100 ; i++)

{

try

{

Thread.sleep(200);

}

catch(InterruptedException e)

{ }

pbar.setValue(i);

pbar.setString(""+i);

}

fr1.setVisible(false);

fr2.setVisible(true);

}

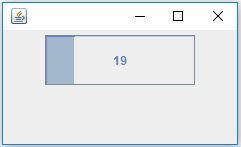
public static void main(String s[])

{

new ProgressTest();

}

}



**To get the current Desktop resolution**

Toolki t=Toolkit.getDefaultToolkit();

Dimention d=t.getScreenSize();

int width=(int)d.getWidth();

int height=(int)d.getHeight();

fr.setSize(Weidth, Height);

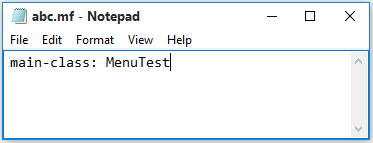
**Create the executable jar of java Application**

**Jar Tool**

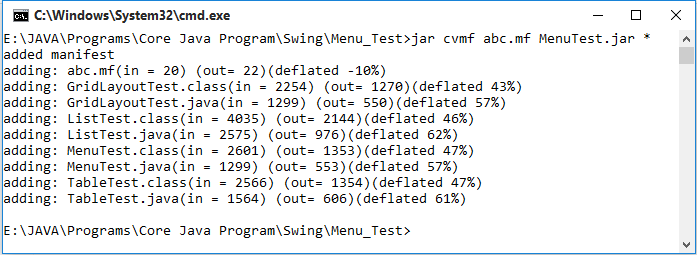
Jar tool is use to make the executable jar file

**Step1: -** Place the **.class** file of java application in any separate folder.

**Step2: -** Create the Menu\_Test file and save it with **.mf** extension in the Menu\_Test file just write the name of main class.



**Step3: -** Open the Dos-Prompt and use the following command **jar** **cvmf**



This executable file will run the public jre.

**APPLET**

Applets are the java that extends the functionality of web page.

Applets program are resides onto the web server and transmitted over the network to be executed on to the web browser.

Applets are the managed objects.

A managed object is the object whose life cycle is managed by the environment.

In case of applet the lifecycle is managed by the web browser life cycle of object means the creation methods onto it and finally description of the object object.

**java.applet.Applet class**

This class provides the life cycle methods that are invoked by the web browser timely onto the applet objet.

**Life-cycle() method is the applet class as follows.**

* + - **public void init()**

This method is invoked by the web browser only once just after the creation of the applet object.

* + - **public void start()**

This method is invoked by the web browser after the init() method and subsequently each time when the applet gets pre loaded.

* + - **public void paint(Graphics g)**

This method always be invoked after the start() method and subsequently each time when the content on the applet redrawn.

* + - **public void stop()**

This method is invoked each time when the execution of the applet gets paused.

* + - **public void destroy()**

This method invoked by the web browser just before the restriction of the applet object.

**Sequence Diagram of Applet lifecycle**

User web Graphics Applet web

Browser object browser

1.1 web page 1.2 request forwarded

request

1.3 HTML page + Applet.class file loaded

1.4 Applet object created

1.5 new object

1.6 init( )

1.7 start( )

1.8 paint( )

2.0 next page

request 2.1 stop( )

3.0 back to

previous page 3.1start,paint( )

4.0 communication

Terminated 4.1 destroy

**[Note: -** All the life-cycle() method are the callback method because their names always be provide by the enviroment.**]**

**//MyApplet.java**

import java.applet.Applet;

import java.awt.\*;

import java.awt.event.\*;

public class MyApplet extends Applet implements ActionListener

{

Button b1,b2;

int x;

public void init()

{

b1=new Button("Rect");

b2=new Button("Circle");

b1.addActionListener(this);

b2.addActionListener(this);

add(b1);

add(b2);

}

public void paint(Graphics g)

{

if(x==1)

{

g.setColor(Color.RED);

g.fillRect(30,70,150,150);

}

if(x==2)

{

g.setColor(Color.GREEN);

g.fillOval(30,70,150,150);

}

}

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==b1)

{

x=1;

}

if(e.getSource()==b2)

{

x=2;

}

repaint();

// to invoke the paint() explicitly

}

}

**[Note: -** Applet class must always be declared as public class.**]**

Applet classes never contains the main() method so the class can’t be run directly through the java tool.

In the web browsers java plug-in are exist to load the Applet class and make the applet objects.

In order to load the Applet class in the web browser just make the html file and make the entry of Applet class into it.

**//MyAppletTest.html**

<html>

<body>

<center>This is my applet</center>

<applet code="MyApplet.class" width="300" height="300">

</applet>

</body>

</html>

**InterApplet Communication**

One Applet can get reference id of another Applet object so that the one Applet can be controlled by another applet.

In web browser there is an object of applet context that contains the reference id of all the applet object created by the java plug-in from the web browser.

Web browser web server

1. request

1.1 html page + Applet class file

downloaded

1.2 Applet 1.3 object of applet

Object are context created Applet Context

created

first Name Ref **1.4** Reference id

Applet First 110 Applet object

Second 220 are stored

Second

Applet

**Method of java.applet.Applet class**

* + - **public AppletContext getAppletContext()**

This method returns the reference id of AppletContext of the object

**Method of AppletContext**

* + - **public Applet getApplet(String appletName)**

This method returns the reference id of applet object whose name specified with the argument.

**[Note: -** Name of the applet has to be specified in the html file within the applet tag.**]**

**Limitation of Applet**

Throuogh the applet program we can never use the user resources outside the boundary of the JRE. That means the applet program we can never access the file from file and database.

**Applet Viewer**

Applet viewer is thetool in the JDK that is use to execute the applet program.

Without making the HTML file we can also run the applet program by making the applets in the .java file entry of the comment.