

Unit-1

Advanced Java Programming

GUI Programming with Java

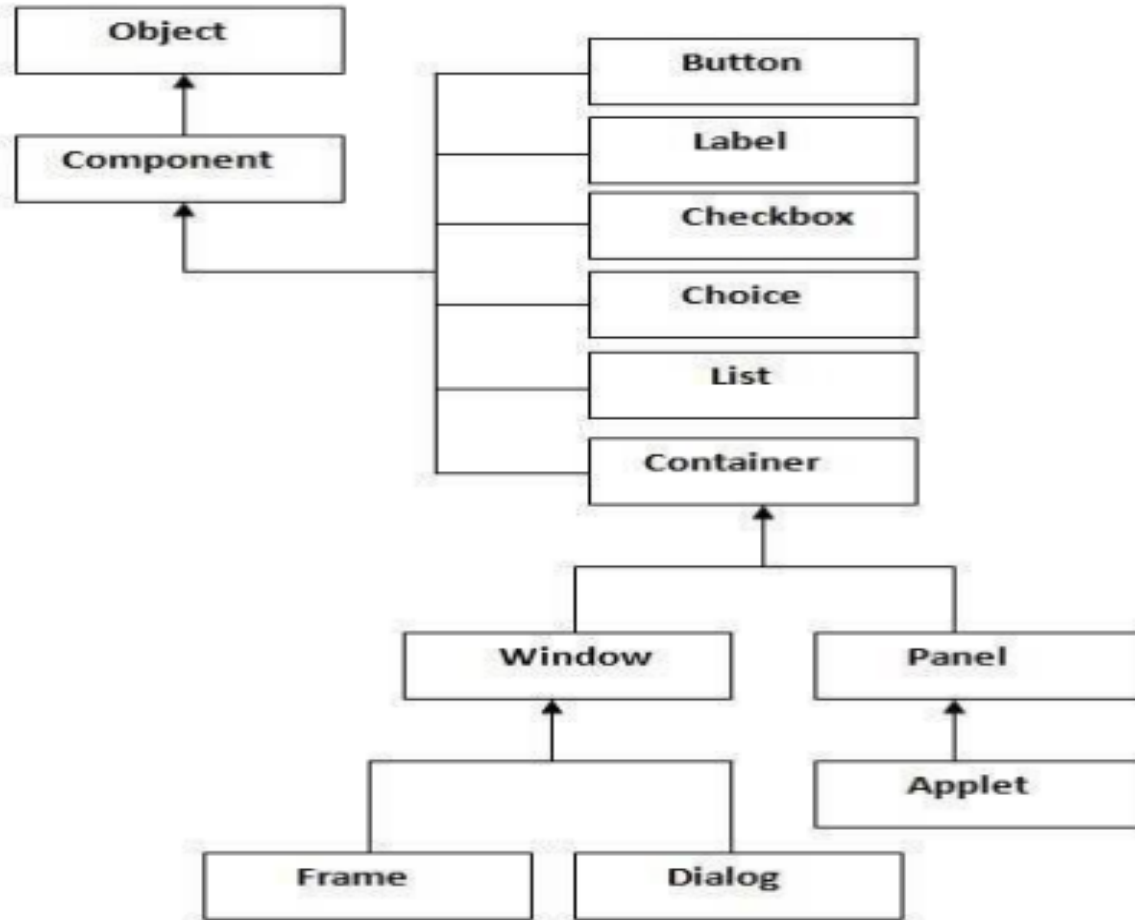
- Graphical user interface is type of user interface that allows user to interact with screen using visual component (Graphical component).
- There are mainly two set of java API for GUI programming.
 - **AWT (Abstract Window Toolkit)**
 - **Swing**

AWT(Abstract Window Toolkit)

- Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc

Java AWT Hierarchy

- The hierarchy of Java AWT classes are given below.



Container

- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc.
The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

- The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

- The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

- The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Java Swing

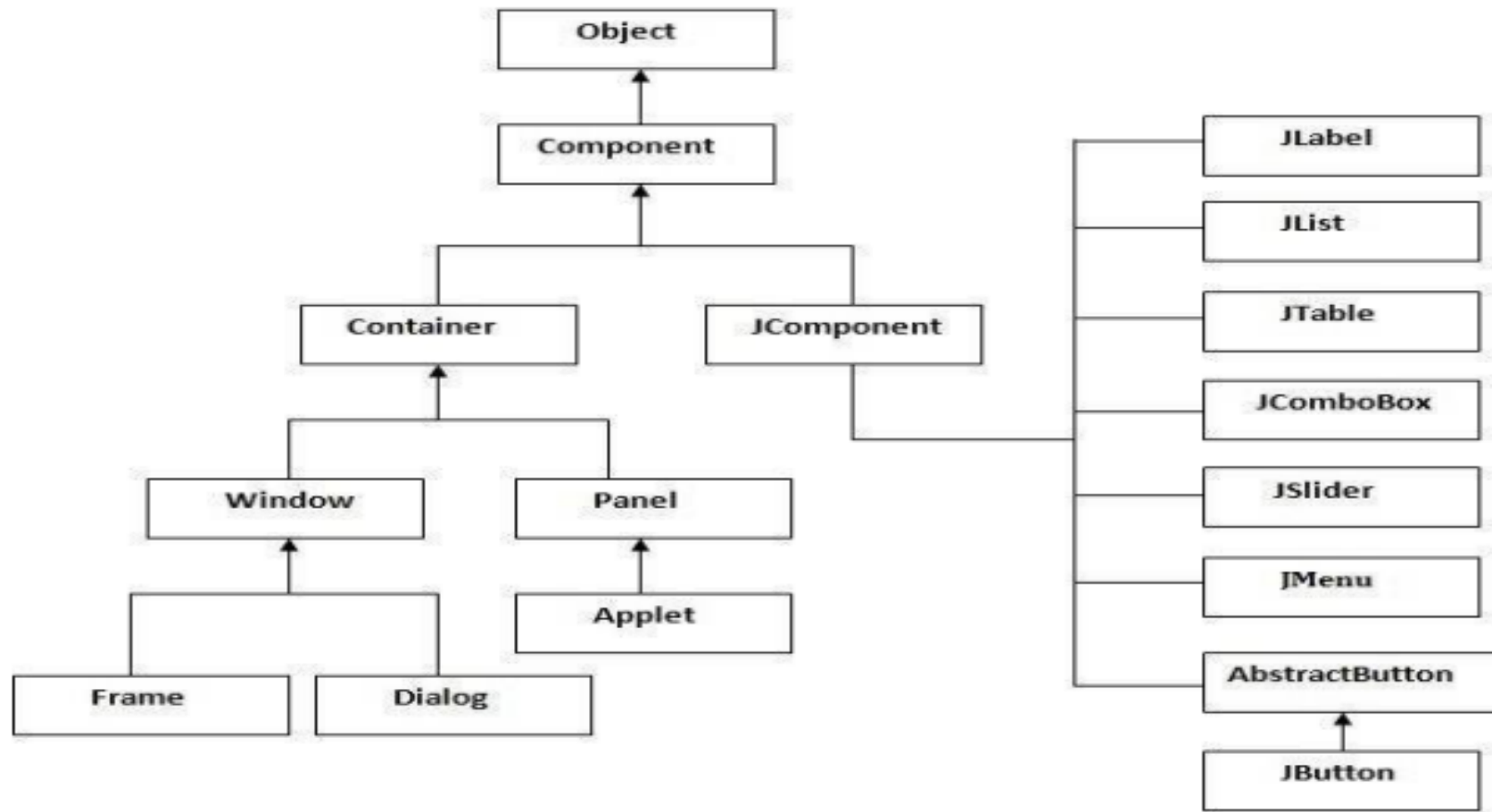
- Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications.
- It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The *javax.swing* package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing.

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent .	Java swing components are platform-independent .
2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC .

Hierarchy of Java Swing classes

- The hierarchy of java swing API is given below.



Commonly used Methods of Component class

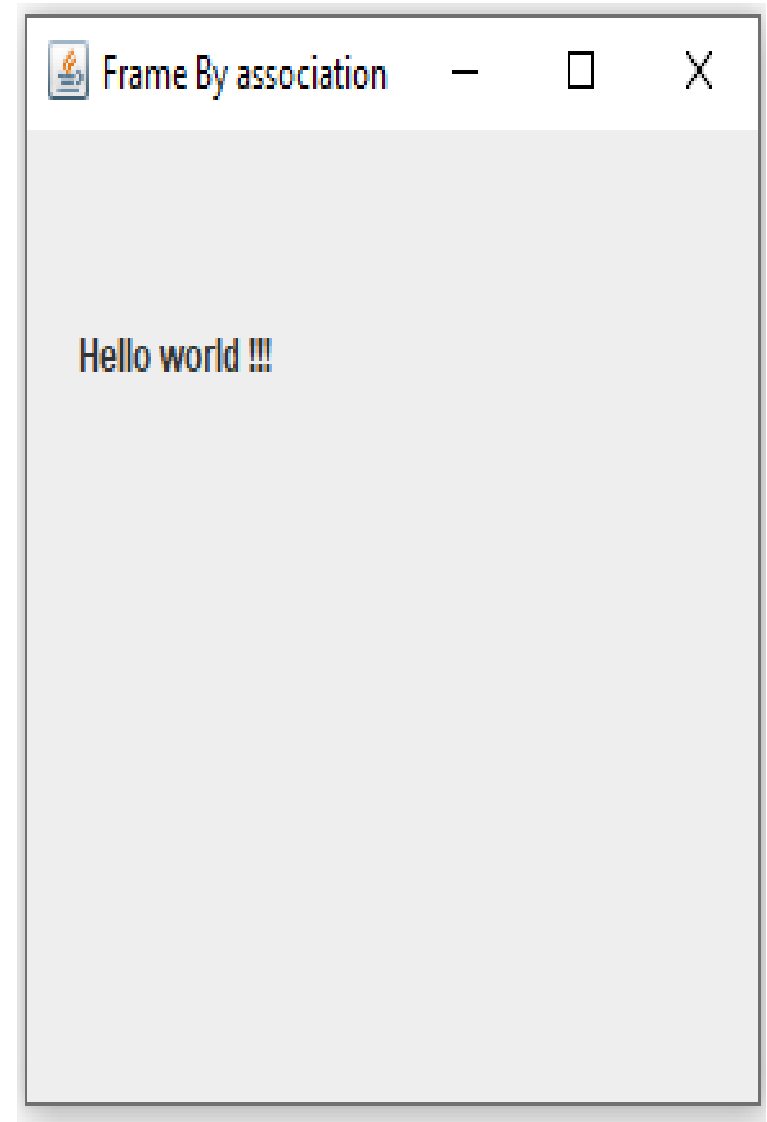
Method	Description
<code>public void add(Component c)</code>	add a component on another component.
<code>public void setSize(int width,int height)</code>	sets size of the component.
<code>public void setLayout(LayoutManager m)</code>	sets the layout manager for the component.
<code>public void setVisible(boolean b)</code>	sets the visibility of the component. It is by default false.

Java JFrame

- The *javax.swing.JFrame* class is a type of container which inherits the *java.awt.Frame* class.
- *JFrame* works like the main window where components like labels, buttons, textfields are added to create a GUI.
- Unlike Frame, JFrame has the option to hide or close the window with the help of `setDefaultCloseOperation(int)` method
- There are two ways to create a frame:
 - **By creating the object of Frame class (association)**
 - **By extending Frame class (inheritance)**

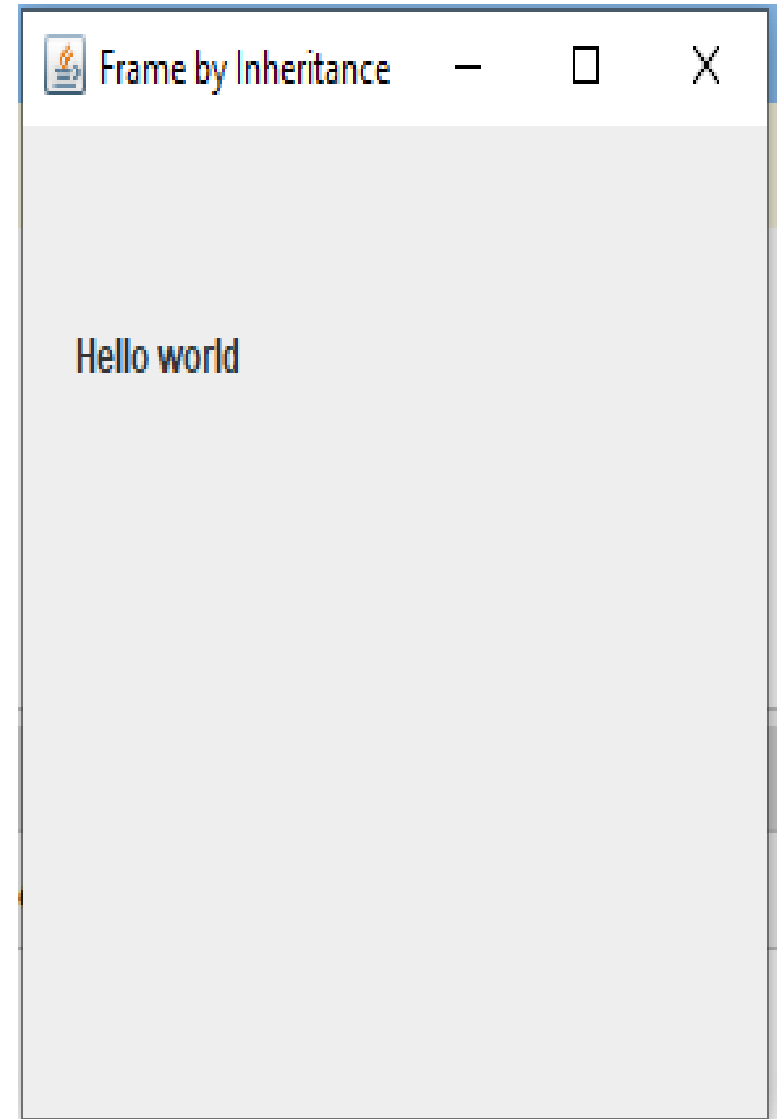
- **Example of JFrame by Association**

```
package FrameEx;
import javax.swing.*.*;
public class FrameEx1 {
    JFrame f;
    JLabel l1;
    public FrameEx1()
    {
        f= new JFrame("Frame By association");
        l1= new JLabel("Hello world !!!");
        f.setSize(300, 300);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLayout(null);
        l1.setBounds(20,50,100,20);
        f.add(l1);
    }
    public static void main(String[] args) {
        new FrameEx1();
    }
}
```



- **Example of JFrame by Inheritance**

```
package FrameEx;
import javax.swing.*.*;
public class FrameEx2 extends JFrame {
    JLabel l1;
    public FrameEx2()
    {
        l1= new JLabel("Hello world");
        setTitle("Frame by Inheritance");
        setSize(300, 300);
        setVisible(true);
        setDefaultCloseOperation(3);
        setLayout(null);
        l1.setBounds(20,50,100,20);
        add(l1);
    }
    public static void main(String[] args) {
        new FrameEx2();
    }
}
```



Layout Manager

- The Layout managers are used to arrange component in a particular manner.
- Layout Manager is an interface that is implemented by all the classes of layout managers.
- A layout manager is an object that controls the size and the position of components in a container.
- Every Container object has LayoutManager object that controls layout
- There are various ways to manage layout, some of them are:
 - **No layout.**
 - **FlowLayout**
 - **BorderLayout**
 - **GridLayout**
 - **GridBagLayout**
 - **Group Layout**

No Layout

- To provide your own explicit layout or absolute positioning of component in a container, following two things can be performed.

- Set the container's layout manager to null by calling `setLayout(null)`

e.g. `frame.setLayout(null);`

- Call the Component class's `setBounds` method for each of the container's children.

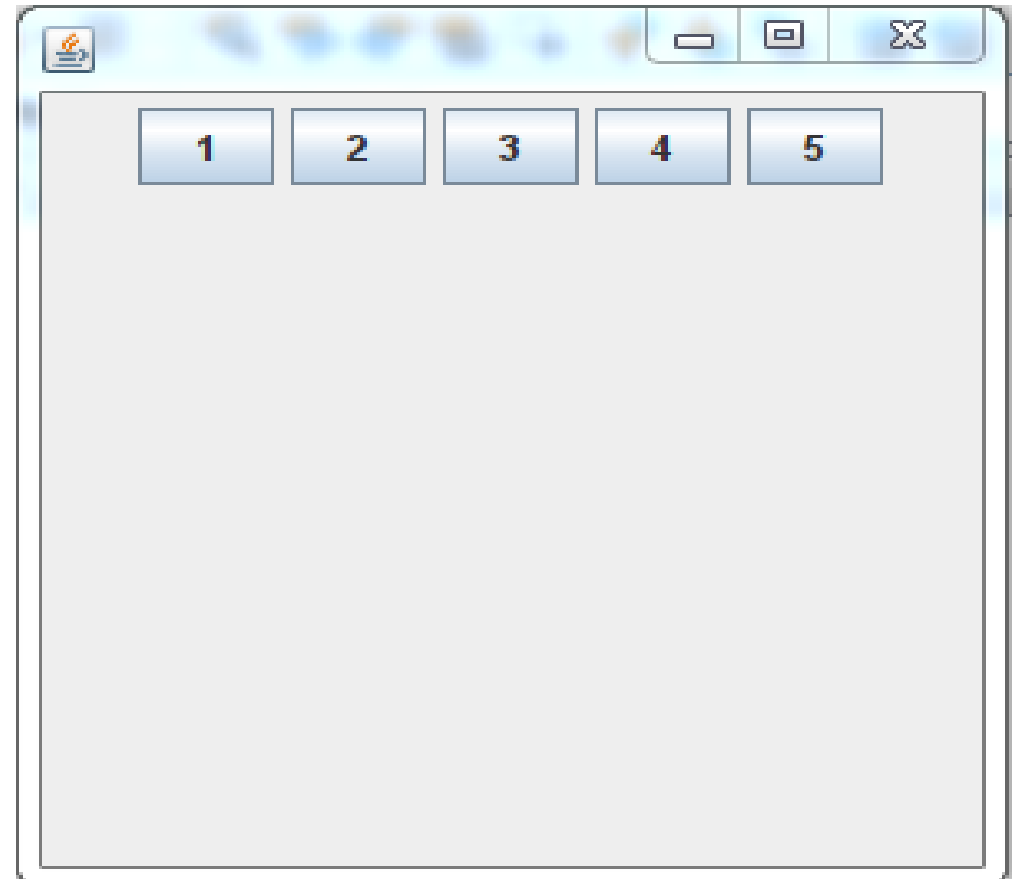
e.g. `component.setBounds(x, y, width, height)`

where (x, y) is the coordinate of the upper-left corner of that component.

The third argument is the **width** of the component and the fourth argument is the **height** of the component.

FlowLayout Manager

- The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow).
- It is the default layout of the applet or panel.
- Constructors of FlowLayout class
 - **FlowLayout()**
 - **FlowLayout(int align)**
 - **FlowLayout(int align, int hgap, int vgap)**



Example:

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

```
public class MyFlowLayout{  
    JFrame f;  
    MyFlowLayout(){  
        f=new JFrame();  
  
        JButton b1=new JButton("1");  
        JButton b2=new JButton("2");  
        JButton b3=new JButton("3");  
        JButton b4=new JButton("4");  
        JButton b5=new JButton("5");
```

```
        // adding buttons to the frame  
        f.add(b1); f.add(b2); f.add(b3); f.add(b  
4); f.add(b5);
```

```
        // setting flow layout  
        f.setLayout(new FlowLayout());
```

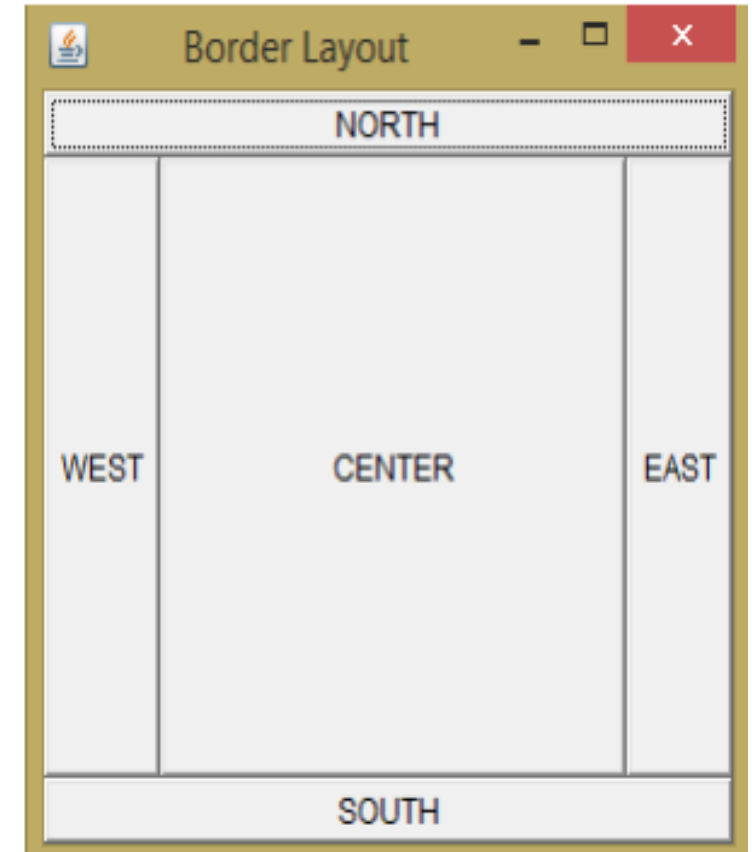
```
        f.setSize(300,300);  
        f.setVisible(true);
```

```
    }
```

```
public static void main(String[] args) {  
    new MyFlowLayout();  
}
```

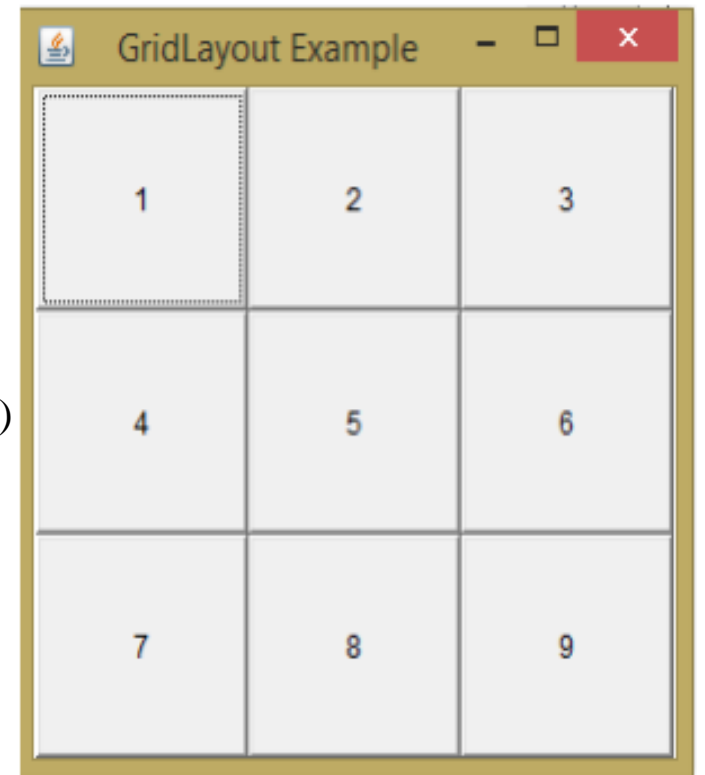
BorderLayout Manager

- The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center.
- Each region (area) may contain one component only.
- The BorderLayout provides five constants for each region:
BorderLayout.CENTER
BorderLayout.SOUTH
BorderLayout.EAST
BorderLayout.WEST
BorderLayout.NORTH
- Here are the constructors defined by BorderLayout:
 - **BorderLayout()**
 - **BorderLayout(int horz, int vert)**



GridLayout Manager

- The Java GridLayout class is used to arrange the components in a rectangular grid.
- When you instantiate a GridLayout, you define the number of rows and columns.
- The constructors supported by GridLayout are shown here:
 - **GridLayout()**
 - **GridLayout(int numRows, int numColumns)**
 - **GridLayout(int numRows, int numColumns, int horz, int vert)**



GUI Control

JLabel:

- The object of ***JLabel*** class is a component for placing text in a container.
- It is used to display a single line of read only text.
- The text can be changed by an application but a user cannot edit it directly.

Constructor:

- JLabel()
- JLabel(String s)
- JLabel(Icon i)
- JLabel(String s,Icon i, int Horizontalalignment)

Methods:

- setText()
- getText()
- setFont(Font f)
- setIcon()
- getIcon()

JTextField:

- The object of a JTextField class is a text component that allows the editing of a single line text

Constructor:

- JTextField()
- JTextField(String s)
- JTextField(int columns)
- JTextField(String s, int columns)

Methods:

- setText()
- getText()
- setFont(Font f)

JTextArea:

- The object of a JTextArea class is a multi line region that displays text.
- It allows the editing of multiple line text.

Constructor:

- JTextArea()
- JTextArea(String s)
- JTextArea(int row, int col)
- JTextField(String s, int row, int col)

Methods:

- setText()
- getText()
- setFont(Font f)
- insert()
- append()

JPasswordField:

- The object of a JPasswordField class is a text component specialized for password entry.
- It allows the editing of a single line of text

Constructor:

- JPasswordField()
- JPasswordField(String s)
- JPasswordField(int columns)
- JPasswordField(String s, int columns)

Methods:

- setText()
- getText()

JButton:

- The JButton class is used to create a labeled button that has platform independent implementation.
- The application result in some action when the button is pushed.

Constructor:

- JButton()
- JButton(String s)
- JButton(Icon i)
- JButton(String s, Icon i)

Methods:

- setText()
- getText()
- setEnabled()

JCheckBox:

- The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false).

Constructor:

- JCheckBox()
- JCheckBox(String s)
- JCheckBox(String s, Boolean selected)

Methods:

- isSelected()
- getItem()
- getItemStateChanged()

Example of JCheckBox

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

public class CheckBoxEx {
    JFrame f;
    JLabel l1;
    JCheckBox c1, c2;

    public CheckBoxEx()
    {
        f= new JFrame("CheckBox Example");
        l1=new JLabel();
        l1.setHorizontalAlignment(JLabel.CENTER);
        l1.setSize(400,100);
        c1 = new JCheckBox("C++");
        c1.setBounds(150,100, 100,50);
        c2 = new JCheckBox("Java");
        c2.setBounds(150,150, 100,50);
        f.add(c1); f.add(c2); f.add(l1);
        c1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                l1.setText("C++ Checkbox: "
                    + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });
    }
}
```

```
    }
});
c2.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        l1.setText("Java Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
    }
});
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}

public static void main(String [] args)
{
    new CheckBoxEx();
}
}
```


JRadioButton:

- The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.
- It should be added in ButtonGroup to select one radio button only.

Constructor:

- JRadioButton()
- JRadioButton(String s)
- JRadioButton(String s, Boolean selected)

Methods:

- setEnabled()
- setText()
- getText()
- setMnemonics()

JComboBox

- Combobox is combination of textfield and dropdown list.
- A Combobox normally display one entry, but it will also have dropdown list that allows user to select different entry.

Constructor:

- JComboBox()
- JComboBox(object [] item)

Methods:

- addItem()
- removeItem()
- setEditable()
- getSelectedItem

JScrollPane

- A JScrollPane is used to make scrollable view of component when screen size is limited.
- JScrollPane is used to display large component or a component whose size can change dynamically.

Constructor:

- JScrollPane()
- JScrollPane(Component)
- JScrollPane(int, int)
- JScrollPane(component, int , int)

Methods:

- setHorizontalScrollBarPolicy()
- setVerticalScrollBarPolicy()

Introduction to Event

- Event is simply user action or Changing the state of an object is known as an event.
- **For example:** click on button, dragging mouse, select item from list, minimizing and maximizing window etc.
- The *java.awt.event* package provides many event classes and Listener interfaces for event handling

Event handling:

- Event handling is mechanism that controls event and decide what should happens if an event occurs.
- In Java, there are two types of event handling
 - **Traditional approach (Java 1.0 model)**
 - **Modern approach (Delegation Event model)**

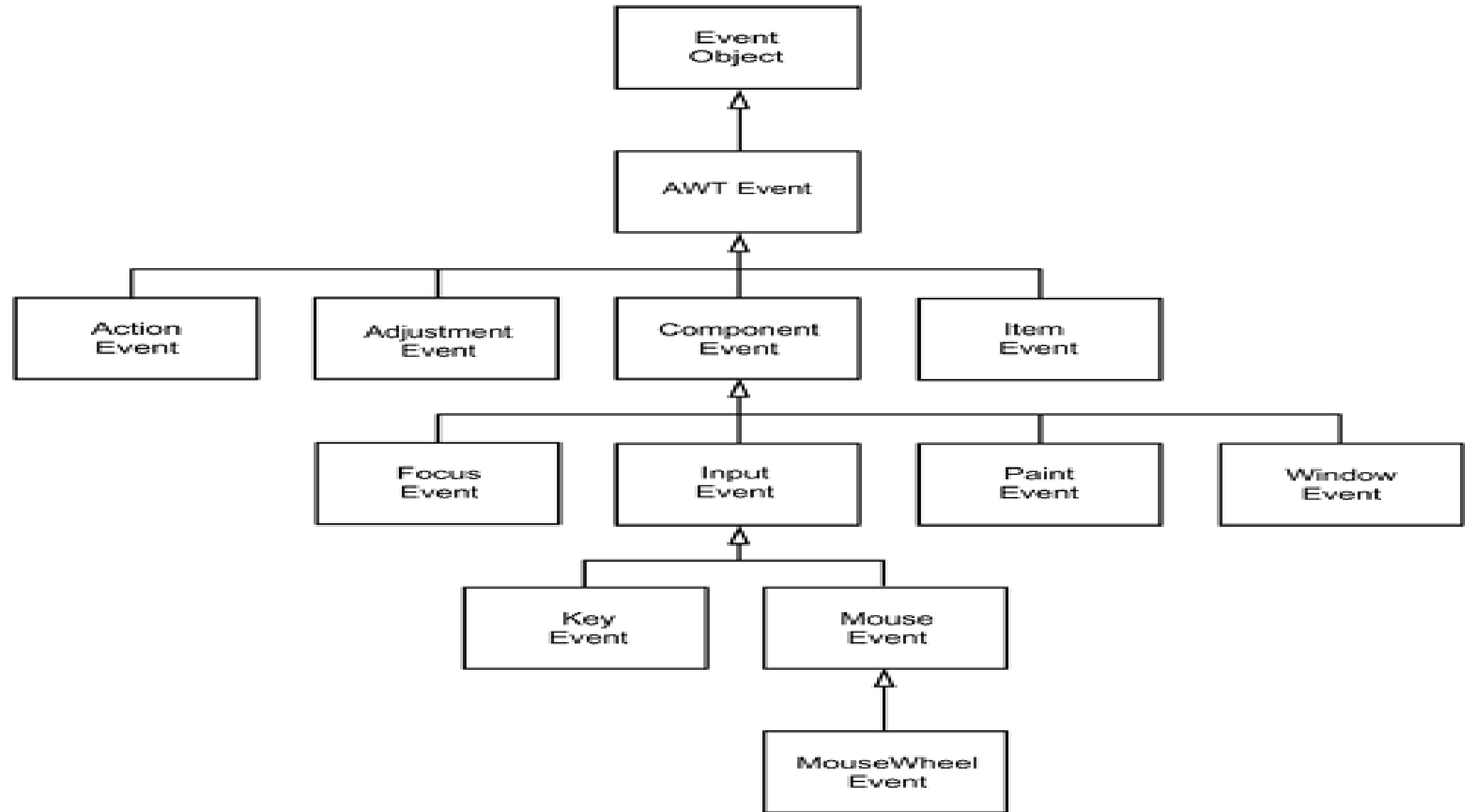
Event Delegation Model:

In Event Delegation model there are mainly two component.

Source: - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.

Listener: - It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event an then returns

AWT event classes



- There are mainly two generic function of event class.
 - **getSource():** returns the source of event.
 - **toString():** returns string equivalent of event

ActionEvent Class

The ActionEvent is generated when button is clicked or the item of a list is double clicked or menu item is clicked.

Method: *String getActionCommand()*

ItemEvent Class

ItemEvent generates when checkbox or a list item is clicked or when checkable menu item is selected or deselected.

Method: *getItem(), getStateChanged()*

KeyEvent class

- KeyEvent generates when input is received from keyboard.
- There are three types of key events which are represented by the integer constants:
KEY_PRESSED, KEY_RELEASED, KEY_TYPED
- Method: *getKeyChar()*, *getKeyCode()*

MouseEvent Class

- MouseEvent generates when mouse is dragged, moved, clicked, pressed, released, and when mouse enter or exit from component.
- Method: *getX()*, *getY()*, *getPoint()*

WindowEvent Class

- The object of this class represents the change in state of a window.
- This event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when focus is transferred into or out of the Window.
- Method: *getNewState()*, *getOldState()*, *getWindow()*

FocusEvent Class

- The FocusEvent is generated when component has gained or lost the input focus
- Method: *isTemporary()*, *paramString()*

Event Listener Interface

- Event listeners represent the interfaces responsible to handle events.
- Java provides various Event listener classes, however, only those which are more frequently used will be discussed.

ActionListener:

- *void actionPerformed(ActionEvent e)*

ItemListener:

- *void itemStateChanged(ItemEvent e)*

KeyListener:

- *void keyTyped(KeyEvent e)*
- *void keyReleased(KeyEvent e)*
- *void keyPressed(KeyEvent e)*

- **MouseListener:**

- *void mouseClicked(MouseEvent e)*
- *void mousePressed(MouseEvent e)*
- *void mouseReleased(MouseEvent e)*
- *void mouseEntered(MouseEvent e)*
- *void mouseExited(MouseEvent e)*

- **MouseMotionListener:**

- *void mouseDragged(MouseEvent e)*
- *void mouseMoved(MouseEvent e)*

- **FocusListener:**

- *void focusGained(FocusEvent e)*
- *void focusLost(FocusEvent e)*

- **WindowListener:**

- *void windowActivated(WindowEvent e)*
- *void windowDeactivated(WindowEvent e)*
- *void windowOpened(WindowEvent e)*
- *void windowClosed(WindowEvent e)*
- *void windowClosing(WindowEvent e)*
- *void windowIconified(WindowEvent e)*
- *void windowDeiconified(WindowEvent e)*

Steps to perform Event Handling

- Following steps are required to perform event handling:
 - Register the component with the Listener
`component.addEventTypeListener(reference)`
 - Implement the Listener interface and overrides its methods
- We can put the event handling code into one of the following places:
 - *Same class*
 - *Other class*
 - *Anonymous class*

Example of event handling within same class:

```
package EventEx;
import javax.swing.*;
import java.awt.event.*;
public class CalculatorEx implements ActionListener {
    JFrame f;
    JLabel l1,l2,l3;
    JTextField t1,t2,t3;
    JButton b1;
    public CalculatorEx()
    {
        f=new JFrame("Calculator");
        l1= new JLabel("First num");
        l2= new JLabel("Second num");
        l3= new JLabel("Result");
        t1= new JTextField(25);
        t2= new JTextField(25);
        t3= new JTextField(25);
        b1= new JButton("Add");

        f.setSize(300,300);
        f.setVisible(true);
        f.setDefaultCloseOperation(3);
        f.setLayout(null);
        l1.setBounds(20,50,100,20);
        t1.setBounds(150,50,100,20);
        l2.setBounds(20,100,100,20);
        t2.setBounds(150,100,100,20);
        b1.setBounds(150,150,80,20);
        l3.setBounds(20,200,100,20);
        t3.setBounds(150,200,100,20);

        f.add(l1);f.add(l2);f.add(l3);
        f.add(t1);f.add(t2);f.add(t3);
        f.add(b1);
        b1.addActionListener(this);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        int fnum=Integer.parseInt(t1.getText());
        int snum=Integer.parseInt(t2.getText());
        int sum=fnum+snum;
        t3.setText(String.valueOf(sum));
    }
    public static void main(String[] args) {
        new CalculatorEx();
    }
}
```

Example of event handling other class:

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

public class CalculatorEx {
    JFrame f;
    JLabel l1,l2,l3;
    JTextField t1,t2,t3;
    JButton b1;

    public CalculatorEx() {
        f=new JFrame("Calculator");
        l1= new JLabel("First num");
        l2= new JLabel("Second num");
        l3= new JLabel("Result");
        t1= new JTextField(25);
        t2= new JTextField(25);
        t3= new JTextField(25);
        b1= new JButton("Add");
        f.setSize(300,300);
        f.setVisible(true);
        f.setDefaultCloseOperation(3);
        f.setLayout(null);
        l1.setBounds(20,50,100,20);
        t1.setBounds(150,50,100,20);
        l2.setBounds(20,100,100,20);
        t2.setBounds(150,100,100,20);
        b1.setBounds(150,150,80,20);

        l3.setBounds(20,200,100,20);
        t3.setBounds(150,200,100,20);

        f.add(l1);f.add(l2);f.add(l3);
        f.add(t1);f.add(t2);f.add(t3);
        f.add(b1);
        b1.addActionListener(new Handler(this));
    }

    public static void main(String[] args) {
        new CalculatorEx();
    }
}

class Handler implements ActionListener
{
    CalculatorEx calc;
    public Handler(CalculatorEx calc)
    {
        this.calc=calc;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        int fnum=Integer.parseInt(calc.t1.getText());
        int snum=Integer.parseInt(calc.t2.getText());
        int sum=fnum+snum;
        calc.t3.setText(String.valueOf(sum));
    }
}
```

Example of event handling using Anonymous class:

```
package EventEx;
import javax.swing.*;
import java.awt.event.*;
public class CalculatorEx {
    JFrame f;
    JLabel l1,l2,l3;
    JTextField t1,t2,t3;
    JButton b1;
    public CalculatorEx()
    {
        f=new JFrame("Calculator");
        l1= new JLabel("First num");
        l2= new JLabel("Second num");
        l3= new JLabel("Result");
        t1= new JTextField(25);
        t2= new JTextField(25);
        t3= new JTextField(25);
        b1= new JButton("Add");

        f.setSize(300,300);
        f.setVisible(true);
        f.setDefaultCloseOperation(3);
        f.setLayout(null);
        l1.setBounds(20,50,100,20);
        t1.setBounds(150,50,100,20);
        l2.setBounds(20,100,100,20);
        t2.setBounds(150,100,100,20);
        b1.setBounds(150,150,80,20);
        l3.setBounds(20,200,100,20);
        t3.setBounds(150,200,100,20);

        f.add(l1);f.add(l2);f.add(l3);
        f.add(t1);f.add(t2);f.add(t3);
        f.add(b1);
        b1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int fnum=Integer.parseInt(t1.getText());
                int snum=Integer.parseInt(t2.getText());
                int sum=fnum+snum;
                t3.setText(String.valueOf(sum));
            }
        });
    }
    public static void main(String[] args) {
        new CalculatorEx();
    }
}
```

Adapter Class

- Adapter class simplify the creation of event handling.
- Java adapter classes *provide the default implementation of listener interface*.
- If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces.
- The adapter classes are found in **java.awt.event**

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

Example: KeyAdapter

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

public class KeyAdapterExample extends KeyAdapter
{
    Label l;
    TextArea area;
    Frame f;

    KeyAdapterExample(){
        f=new Frame("Key Adapter");
        l=new Label();
        l.setBounds(20,50,200,20);
        area=new TextArea();
        area.setBounds(20,80,300, 300);
        area.addKeyListener(this);
        f.add(l);f.add(area);
```

```
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }

    public void keyReleased(KeyEvent e) {
        String text=area.getText();
        l.setText(" Total enetered
characters:"+text.length());
    }

    public static void main(String[] args) {
        new KeyAdapterExample();
    }
}
```

Menu

- The JMenuBar class is used to display menubar on the window or frame. It may have several menus.
- The JMenuBar class provides an implementation of a menu bar. For creating menu we use JMenuItem and JMenu classes

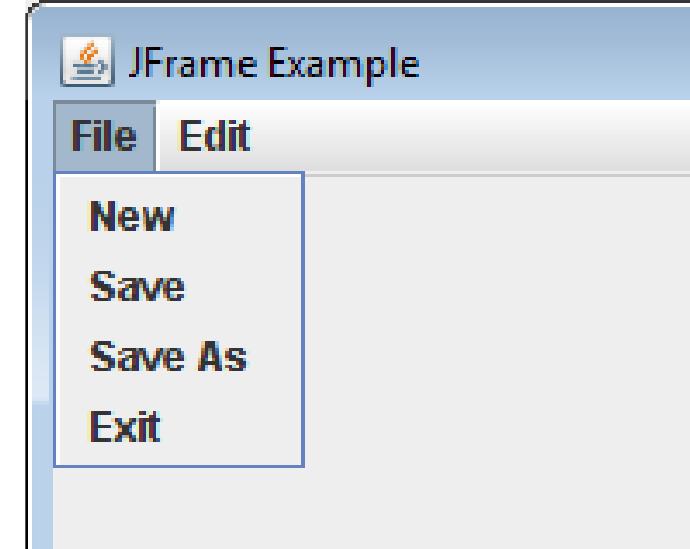
Constructor:

- **JMenuBar()**
- **JMenu** encapsulates a menu, which is populated with **JMenuItems**
 - **JMenu();**
 - **JMenu(String name);**
- **JMenuItem** encapsulates an element in a menu
 - **JMenuItem(String *name*)**
 - **JMenuItem(Icon *image*)**
 - **JMenuItem(String *name*, Icon *image*)**
 - **JMenuItem(String *name*, int *mnem*)**

Example: **menu creation in swing**

```
import javax.swing.*;  
import java.awt.*;  
class Abc extends JFrame  
{  
    JMenuBar jmb;  
    JMenu m1,m2;  
    JMenuItem i1,i2,i3,i4;  
    Abc()  
    {  
        jmb=new JMenuBar();  
        m1=new JMenu("File");  
        m2=new JMenu("Edit");  
        jmb.add(m1);  
        jmb.add(m2);
```

```
i1=new JMenuItem("New");  
i2=new JMenuItem("Save");  
i3=new JMenuItem("Save As");  
i4=new JMenuItem("Exit");  
m1.add(i1);  
m1.add(i2);  
m1.add(i3);  
m1.add(i4);  
setLayout(new FlowLayout());  
setTitle("JFrame Example");  
setSize(300,300);  
setJMenuBar(jmb);  
setVisible(true);  
}  
}  
public static void main(String ar[])  
{  
    new Abc();  
}  
}
```



Use **JRadioButtonMenuItem** and **JCheckBoxMenuItem**

- Swing defines two other menu items: check boxes and radio buttons.
- To add a check box to a menu, create a **JCheckBoxMenuItem**

Constructors

- `JCheckBoxMenuItem(String name)`
- `JCheckBoxMenuItem(String name, boolean state)`
- `JCheckBoxMenuItem(String name, Icon icon)`
- A radio button can be added to a menu by creating an object of type

JRadioButtonMenuItem

Constructors:

- `JRadioButtonMenuItem(String name)`
- `JRadioButtonMenuItem(String name, boolean state)`
- `JRadioButtonMenuItem(String name, Icon icon, boolean state)`

Example:Jcheckboxmenuitem and jradiobuttonmenuitem

```
import java.awt.*;
import javax.swing.*;
public class SwingEx {
    JFrame f1;
    JMenuBar br;
    JMenu mn;
    JCheckBoxMenuItem c1,c2;
    JRadioButtonMenuItem r1,r2;
    SwingEx()
    {
        f1=new JFrame("MenuExample");
        br=new JMenuBar();
        mn=new JMenu("File");
        c1= new JCheckBoxMenuItem("item1");
        c2= new JCheckBoxMenuItem("item2");
        r1= new JRadioButtonMenuItem("ritem1");
        r2= new JRadioButtonMenuItem("ritem2");
```

```
ButtonGroup bg=new ButtonGroup();
    bg.add(r1);
    bg.add(r2);
    mn.add(c1);
    mn.add(c2);
    mn.add(r1);
    mn.add(r2);
    br.add(mn);
    f1.setJMenuBar(br);
    f1.setSize(400,400);
    f1.setLayout(new FlowLayout());
    f1.setVisible(true);

}
public static void main(String []args)
{
    new SwingEx();
}

}
```

PopUp menu

- Popup menu represents a menu which can be dynamically popped up at a specified position within a component.

Constructors:

- JPopupMenu()
- JPopupMenu(String label)

Example:


```
import javax.swing.*;
import java.awt.event.*;
class PopupMenuExample
{
    PopupMenuExample(){
        JFrame f= new JFrame("PopupMenu Example");
        JPopupMenu popupmenu = new JPopupMenu("Edit");
        JMenuItem cut = new JMenuItem("Cut");
        JMenuItem copy = new JMenuItem("Copy");
        JMenuItem paste = new JMenuItem("Paste");
        popupmenu.add(cut);
        popupmenu.add(copy);
        popupmenu.add(paste);
    }
}
```

```
f.addMouseListener(new MouseAdapter() {  
    public void mouseClicked(MouseEvent e) {  
        popupmenu.show(f , e.getX(), e.getY());  
    }  
});  
f.add(popupmenu);  
f.setSize(300,300);  
f.setLayout(null);  
f.setVisible(true);  
}  
public static void main(String args[])  
{  
    new PopupMenuExample();  
}}
```

Add Mnemonics and Accelerators to Menu Items

- In real applications, a menu usually includes support for keyboard shortcuts because they give an experienced user the ability to select menu items rapidly
- Keyboard shortcuts come in two forms: mnemonics and accelerators.
- a *mnemonic* defines a key that lets you select an item from an active menu by typing the key.
- An *accelerator* is a key that lets you select a menu item without having to first activate the menu.
- There are two ways to set the mnemonic for **JMenuItem**.
 - `JMenuItem(String name, int mnem)` **(first way)**
 - `void setMnemonic(int mnem)` **(second way)**

- *mnem* specifies the mnemonic. such as **KeyEvent.VK_F** or **KeyEvent.VK_Z**.
- An accelerator can be associated with a **JMenuItem** object. It is specified by calling **setAccelerator()**
 - void setAccelerator(**KeyStroke** *ks*)
 - static KeyStroke getKeyStroke(int *ch*, int *modifier*)
- The value of *modifier* must be one or more of the following constants.
 - **InputEvent.CTRL_DOWN_MASK**
 - **InputEvent.ALT_DOWN_MASK**
 - **InputEvent.SHIFT_DOWN_MASK**

- **Example:**

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class MnemonicExample {
    JFrame f;
        JMenuBar jb;
        JMenu m1, m2 ;
        JMenuItem i1, i2;

MnemonicExample() {
    f=new JFrame("MnemonicExample");
        jb=new JMenuBar();
        m1=new JMenu ("File");
        m2=new JMenu("Edit");
    i1=new JMenuItem("New");
    i1.setMnemonic(KeyEvent.VK_N);
    i1.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,KeyEvent.CTRL_MASK));
        i2=new JMenuItem("Open");
    i2.setMnemonic(KeyEvent.VK_O);
    i2.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,KeyEvent.ALT_MASK));
```

```
f.setJMenuBar(jb);
jb.add(m1);jb.add(m2); m1.add(i1);m1.add(i2);
f.setSize(400,400);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
i1.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e)
    {
        JFrame f1= new JFrame("New Frame");
        f1.setSize(200,200);
        f1.add(new JLabel("new menu item is clicked"));
        f1.setVisible(true);
        f1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
});
}

public static void main(String [] args)
{
    new MnemonicExample();
}
}
```

using ToolTip in Java Swing

➤ To create a tool tip for any Jcomponent with **setToolTipText()** method.

➤ **Jcomponent.setToolTipText(string str);**

➤ **Example:**

```
import java.awt.*;
```

```
import javax.swing.*;
```

```
public class TooltipExample {
```

```
    JFrame f;
```

```
    JPasswordField pf;
```

```
    JLabel lb;
```

```
    TooltipExample() {
```

```
        f=new JFrame("Tooltip Example");
```

```
        pf= new JPasswordField(15);
```

```
pf.setToolTipText("Enter your password");  
  
    lb= new JLabel("Password:");  
    f.add(lb);  
    f.add(pf);  
    f.setSize(400,400);  
    f.setVisible(true);  
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    f.setLayout(new FlowLayout());  
  
}  
  
public static void main(String [] args)  
{  
    new TooltipExample();  
}  
}
```



Using JToolBar in java swing

- A toolbar is a component that can serve as both an alternative and as an adjunct to a menu.
- A toolbar contains a list of buttons (or other components) that give the user immediate access to various program options
- **Constructor**
 - JToolBar()
 - JToolBar(String name)
 - JToolBar(int orientation)
 - JToolBar(String name, int orientation)

Example:

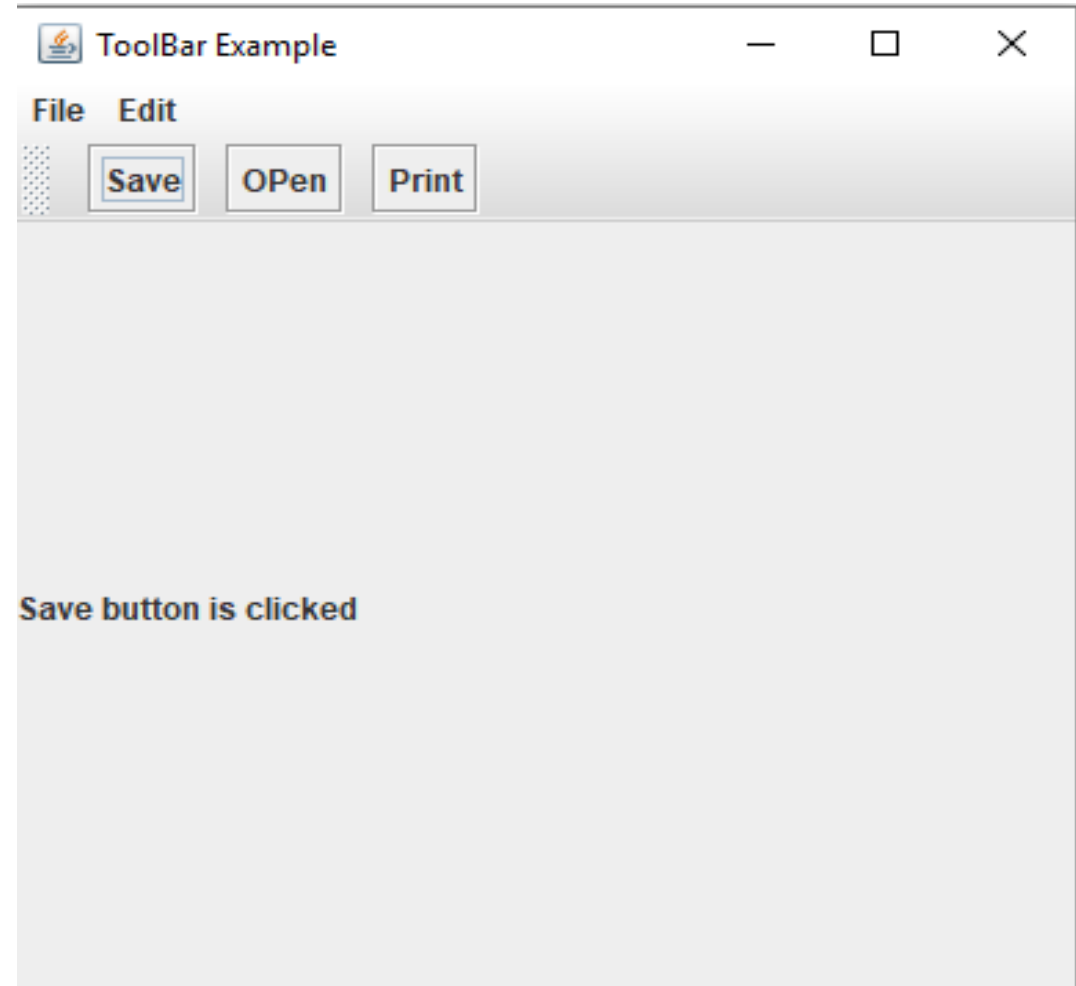
```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
public class ToolBarExample {  
    JFrame f;  
    JMenuBar br;  
    JMenu m1,m2;  
    JToolBar tb;  
    JButton b1, b2, b3;  
    JLabel lb;  
    ToolBarExample() {  
        f= new JFrame("ToolBar Example");  
        br= new JMenuBar();  
        m1=new JMenu("File");  
        m2=new JMenu("Edit");  
        tb= new JToolBar("File Toolbar");  
        b1= new JButton("Save");  
        b2= new JButton("Open");  
        b3= new JButton("Print");
```

```

tb.addSeparator();
br.add(m1);br.add(m2);
tb.add(b1);tb.addSeparator();tb.add(b2);tb.addSeparator();tb.add(b3);
f.setSize(400,400);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setJMenuBar(br);
f.add(tb, BorderLayout.NORTH);
b1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        lb=new JLabel("Save button is clicked");
        f.add(lb, BorderLayout.CENTER);
    }
});
}

public static void main(String [] arg)
{
    new ToolBarExample();
}
}

```



Dialog Box Using Swing

- Dialog boxes are windows in which programs display important messages to the user or obtain information from the user
- Java's **JOptionPane** class (package **javax.swing**) provides prebuilt dialog boxes for both input and output





Input Dialogs

- **JOptionPane** static method **showInputDialog** is used to displays an input dialog

Message Dialogs

- **JOptionPane** static method **showMessageDialog** is used to display a message dialog

JOptionPane Message Dialog Constants

Message dialog type	Icon	Description
ERROR_MESSAGE		Indicates an error.
INFORMATION_MESSAGE		Indicates an informational message.
WARNING_MESSAGE		Warns of a potential problem.
QUESTION_MESSAGE		Poses a question. This dialog normally requires a response, such as clicking a Yes or a No button.
PLAIN_MESSAGE	no icon	A dialog that contains a message, but no icon.

Example:

```
import javax.swing.*;

public class OptionPaneExample {

    public static void main(String [] arg)

    {

        int fnum=Integer.parseInt(JOptionPane.showInputDialog("Enter First Number"));

        int snum=Integer.parseInt(JOptionPane.showInputDialog("Enter Second Number"));

        int sum= fnum+snum;

        JOptionPane.showMessageDialog(null,"Sum of two num="+sum, "sum of two integer",
JOptionPane.PLAIN_MESSAGE);

    }

}
```

Creating Dialogs

- The JDialog control represents a top level window with a border and a title used to take some form of input from the user.
- Unlike JFrame, it doesn't have maximize and minimize buttons.

Constructor:

- JDialog()
- JDialog(Frame owner)
- JDialog(Frame owner, String title, boolean modal)

Example:

```
import javax.swing.*;
```

```
import java.awt.*;
```

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

```
public class DialogExample {
```

```
    private static JDialog d;
```

```
    DialogExample() {
```

```
        JFrame f= new JFrame();
```

```
        d = new JDialog(f , "Dialog Example", true);
```

```
        d.setLayout( new FlowLayout() );
```

```
        JButton b = new JButton ("OK");
```



```
b.addActionListener ( new ActionListener()  
    {  
        public void actionPerformed((ActionEvent e )  
        {  
            DialogExample.d.setVisible(false);  
        }  
    });  
d.add( new JLabel ("Click button to continue."));  
d.add(b);  
d.setSize(300,300);  
d.setVisible(true);  
}  
public static void main(String args[])  
{  
    new DialogExample();  
}  
}
```



Color Choosers Dialog Box

➤ The JColorChooser class is used to create a color chooser dialog box so that user can select any color.

Constructors:

- JColorChooser (): is used to create a color chooser pane with white color initially.
- JColorChooser (Color c): is used to create a color chooser pane with the specified color initially.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JcolorChooserExample {
    JFrame f;
    JButton bt;
    Container c1;

    public JcolorChooserExample() {
        f=new JFrame("Jcolor chooser Example");
        bt= new JButton("Color");
        c1=f.getContentPane();
        c1.add(bt);
        f.setSize(400,400);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

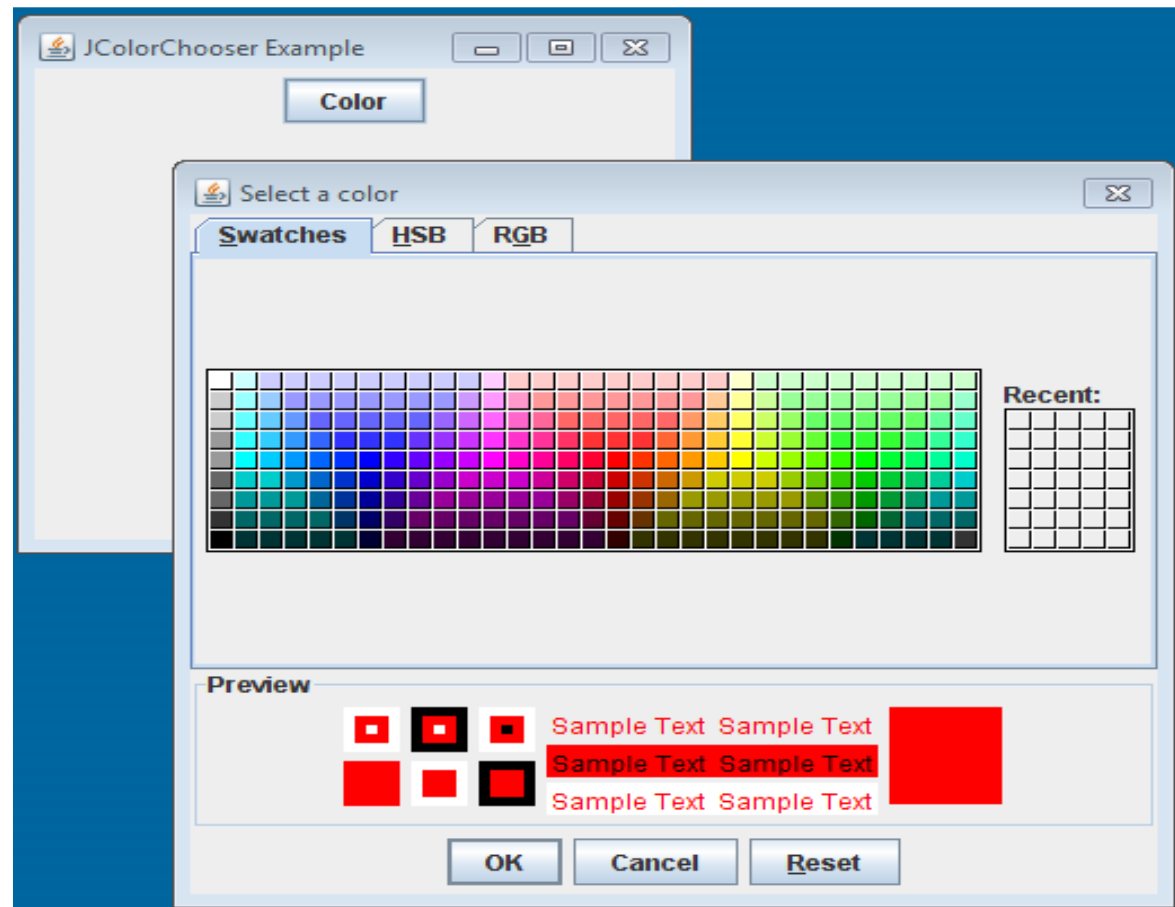
```

f.setLayout(new FlowLayout());
    bt.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            Color incolor=Color.RED;
            Color c= JColorChooser.showDialog(f, "Jcolorchooser Example", incolor);
            c1.setBackground(c);
        }
    });

}

public static void main(String [] arg)
{
    new JcolorChooserExample();
}
}

```



JFileChooser dialog

- FileChooser is a easy and an effective way to prompt the user to choose a file or a directory
- The object of JFileChooser class represents a dialog window from which the user can select file.

Constructor

- JFileChooser()
- JFileChooser(File currentDirectory)
- JFileChooser(String currentDirectoryPath)

Example: source code

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
import java.io.*;  
public class fileChooserExample {  
    JFrame f;  
    JMenuBar br;  
    JMenu m1;  
    JMenuItem opn;  
    JTextArea ta;  
  
    public fileChooserExample() {  
        f= new JFrame("File chooser example");  
        br= new JMenuBar();  
        m1= new JMenu("File");  
        opn=new JMenuItem("Open");  
        ta=new JTextArea(50,50);  
        m1.add(opn);br.add(m1);  
        f.add(ta);
```

```

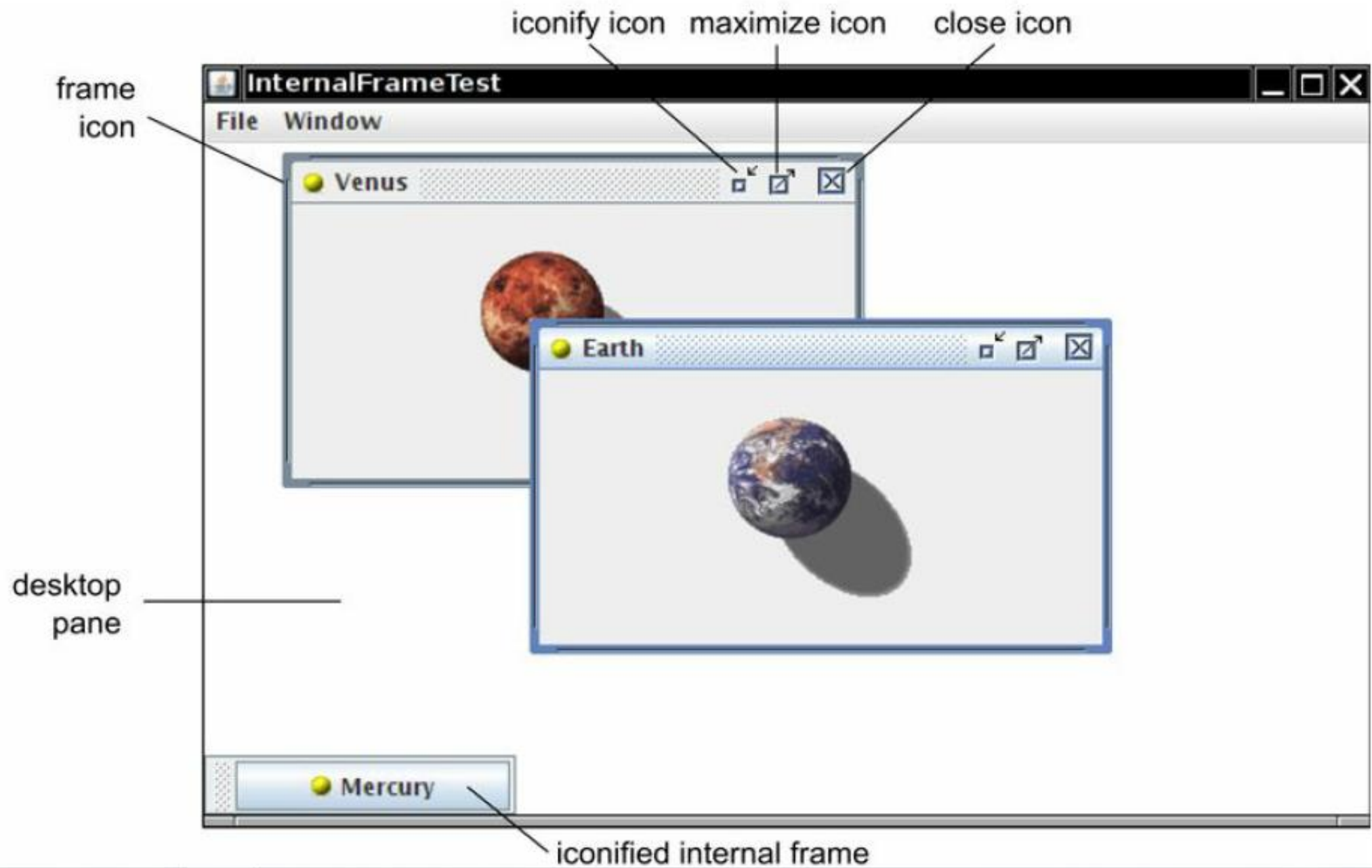
f.setJMenuBar(br);
    f.setSize(400,400);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);
    f.setLayout(new FlowLayout());
    opn.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if(e.getSource()== opn)
            {
                JFileChooser fc= new JFileChooser();
                int i=fc.showOpenDialog(f);
                if(i==JFileChooser.APPROVE_OPTION)
                {
                    File f=fc.getSelectedFile();
                    String fpath=f.getPath();
                    try{
                        BufferedReader br= new BufferedReader(new
FileReader(fpath));
                        String str1=" ",str2=" ";
                        while((str1=br.readLine())!=null)
                        {
                            str2+=str1+"\n";
                        }
                    }
                    ta.setText(str2);
                    br.close();
                }catch(Exception ex)
                {
                    ex.printStackTrace();
                }
            }
        }
    });
}

public static void main(String [] args)
{
    new fileChooserExample();
}

```

Desktop Panes and Internal Frames

- Many applications present information in multiple windows that are all contained inside a large frame.
- In the Windows environment, this user interface is sometimes called the *multiple document interface*(MDI).
- The JDesktopPane class, can be used to create "multi-document" applications.
- To make multi-document app, get contentPane in the main window as an instance of the JDesktopPane class or a subclass.
- Internal windows add instances of JInternalFrame to the JdesktopPane instance.



Constructor For JDesktopPane:

- **JDesktopPane();**

Constructor For JFrame:

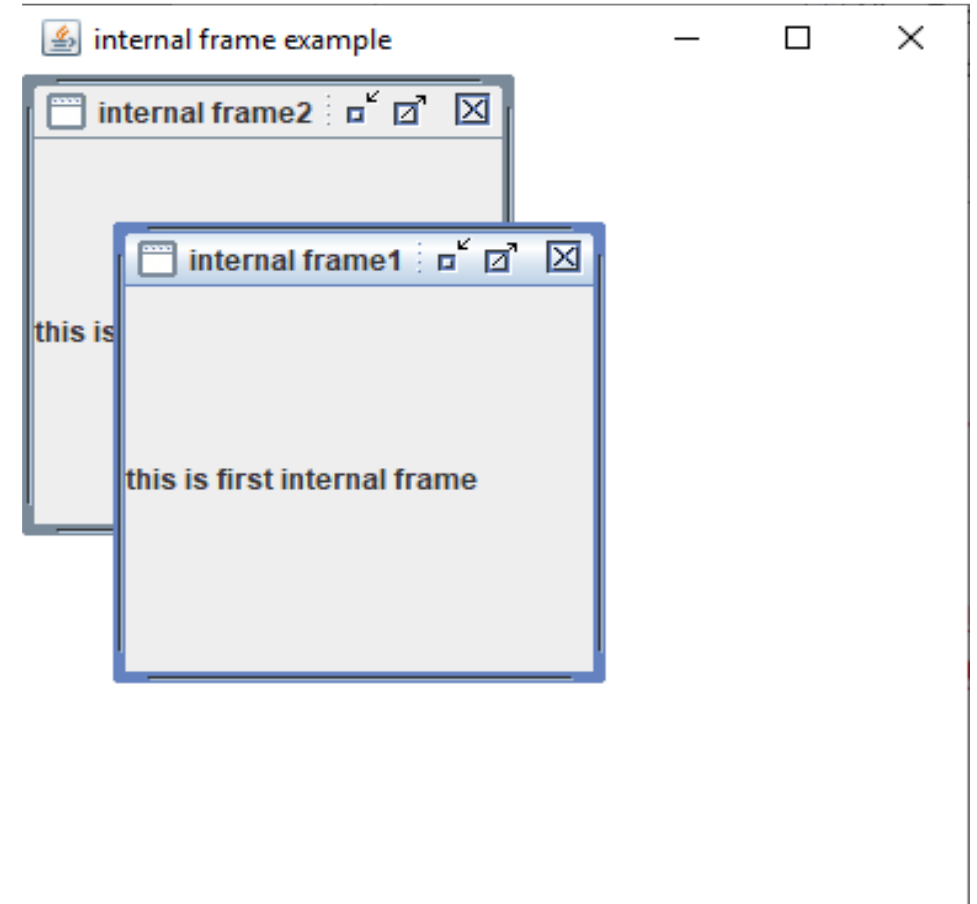
- **JFrame()**
- **JFrame(String t)**
- **JFrame(String t, boolean resizable, boolean closable, boolean maximizable, boolean iconifiable)**

Example:

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

public class internalframeexample {
    JFrame f;
    JDesktopPane dp;
    JInternalFrame f1,f2;

    public internalframeexample() {
        f= new JFrame("internal frame example");
        dp=new JDesktopPane();
        f.add(dp);
        f1=new JInternalFrame("internal frame1", true, true, true, true);
        f2=new JInternalFrame("internal frame2", true, true, true, true);
```



```
f1.setSize(200,200);
    f2.setSize(200,200);
    f1.setVisible(true);
    f2.setVisible(true);
    f1.getContentPane().add(new JLabel("this is first internal frame"));
    f2.getContentPane().add(new JLabel("this is second internal frame"));
    dp.add(f1);dp.add(f2);
    f.setSize(400,400);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

}

public static void main(String [] arg)
{
    new interalframexample();
}
}
```

List

- A list displays a series of items from which the user may select one or more items.
- Lists are created with class **Jlist**.
- Class JList supports **single selection lists** (which allow only one item to be selected at a time) and **multiple-selection lists**(which allow any number of items to be selected).

Constructor:

- **JList()**
- **JList(arr[] data element)**

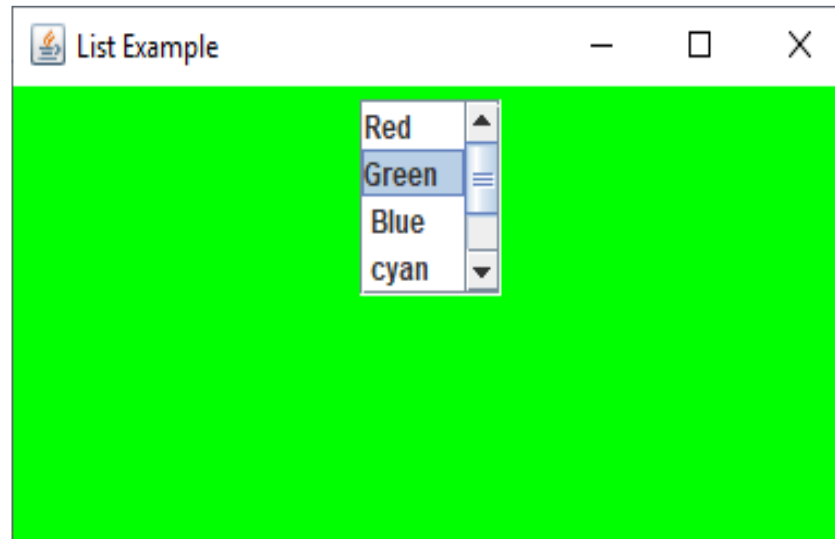
Example:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
public class ListExample {
    JFrame f;
    JList list;

    public ListExample() {
        f= new JFrame("List Example");
        f.setLayout(new FlowLayout());
        String [] listcol={"Red","Green"," Blue"," cyan","Black ","Orange "};
        Color [] color={Color.red,Color.green,Color.blue,Color.cyan,Color.black,Color.orange};
        list=new JList(listcol);
        list.setVisibleRowCount(4);
    }
}
```

```
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
f.add(new JScrollPane(list));
f.setSize(400,400);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
list.addListSelectionListener(new ListSelectionListener() {
    @Override
    public void valueChanged(ListSelectionEvent e) {
        f.getContentPane().setBackground(color[list.getSelectedIndex()]);
    }
});
}

public static void main(String [] arg)
{
    new ListExample();
}
}
```



JTable

- A table is a component that displays data in tabular form. It is composed of rows and columns.
- Tables are implemented by the **JTable class**, which extends JComponent.

Constructor:

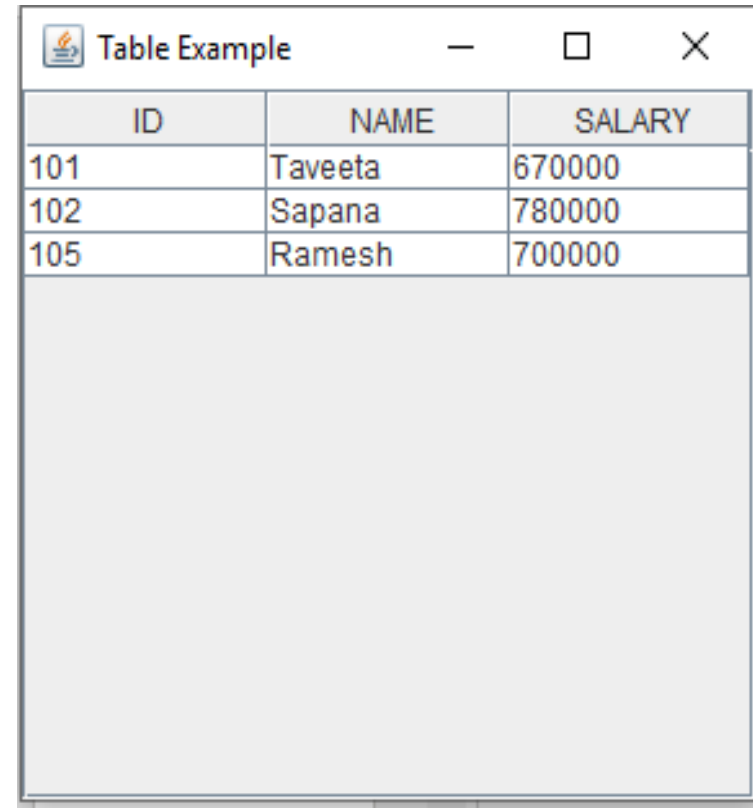
- **JTable();**
- **JTable(Object data[][], Object colHeads[]);**
- Here are the steps for using a table in a Frame:
 - **Create a JTable object.**
 - **Create a JScrollPane object.**
 - **Add the table to the scroll pane.**
 - **Add the scroll pane to JFrame**

Example:

```
import javax.swing.*;

public class TableExample {
    JFrame f;
    JTable tb;
    JScrollPane js;

    TableExample() {
        f=new JFrame("Table Example");
        String data[][]={ {"101","Taveeta","670000"},
                           {"102","Sapana","780000"},
                           {"105","Ramesh","700000"} };
        String column[]={ "ID","NAME","SALARY" };
        tb=new JTable(data, column);
        js=new JScrollPane(tb);
```



ID	NAME	SALARY
101	Taveeta	670000
102	Sapana	780000
105	Ramesh	700000

```
f.add(js);  
    f.setSize(300,300);  
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    f.setVisible(true);  
}
```

```
public static void main(String [] args)  
{  
    new TableExample();  
}  
  
}
```

Working With 2D Shapes

- The AWT includes several methods that support graphics.
- All graphics are drawn relative to a window.
- The origin of each window is at the top-left corner and is 0,0. Coordinates are specified in pixels.
- All output to a window takes place through a graphics context.
- A graphics context is encapsulated by the Graphics class.
- Here are two ways in which a graphics context can be obtained:
 - *It is passed to a method, such as paint() or update(), as an argument.*
 - *It is returned by the getGraphics() method of Component.*
- Among other things, the Graphics class defines a number of methods that draw various types of objects, such as lines, rectangles, and arcs. In several cases, objects can be drawn edge-only or filled
- Objects are drawn and filled in the currently selected color, which is black by default.

- There are several ways to create graphics in Java; the simplest way is to use *java.awt.Canvas* and *java.awt.Graphics*.
- A Canvas is a blank rectangular area of the screen onto which the application can draw.
- The Graphics class provides basic drawing methods such as drawLine, drawRect, and drawString.

```
import java.awt.Canvas;
import java.awt.Graphics;
import javax.swing.JFrame;
public class Drawing extends Canvas {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My Drawing");
        Canvas canvas = new Drawing();
        canvas.setSize(400, 400);
        frame.add(canvas);
        frame.pack();
        frame.setVisible(true);
    }
    public void paint(Graphics g) {
        g.fillRect(100, 100, 200, 200);
    }
}
```

- The Drawing class extends Canvas, so it has all the methods provided by Canvas, including setSize.
- In the main method, we
 - *Create a JFrame object, which is the window that will contain the canvas.*
 - *Create a Drawing object (which is the canvas), set its width and height, and add it to the frame.*
 - *Pack the frame (resize it) to fit the canvas, and display it on the screen.*
- Once the frame is visible, the paint method is called whenever the canvas needs to be drawn; for example, when the window is moved or resized

Drawing Lines

- Lines are drawn by means of the drawLine() method, shown here:
 - *void drawLine(int startX, int startY, int endX, int endY)*
- drawLine() displays a line in the current drawing color that begins at startX, startY and ends at endX, endY

Drawing Rectangles

- The drawRect() and fillRect() methods display an outlined and filled rectangle, respectively.
- They are shown here:
 - *void drawRect(int left, int top, int width, int height)*
 - *void fillRect(int left, int top, int width, int height)*
- To draw a rounded rectangle, use drawRoundRect() or fillRoundRect(), both shown here:
 - *void drawRoundRect(int left, int top, int width, int height, int xDiam, int yDiam)*
 - *void fillRoundRect(int left, int top, int width, int height, int xDiam, int yDiam)*

Drawing Ellipses and Circles

- To draw an ellipse, use `drawOval()`. To fill an ellipse, use `fillOval()`. These methods are shown here:
 - *`void drawOval(int left, int top, int width, int height)`*
 - *`void fillOval(int left, int top, int width, int height)`*
- The ellipse is drawn within a bounding rectangle whose upper-left corner is specified by `left`, `top` and whose width and height are specified by `width` and `height`.
- To draw a circle, specify a square as the bounding rectangle.

Drawing Arcs

- Arcs can be drawn with `drawArc()` and `fillArc()`, shown here:
 - *`void drawArc(int left, int top, int width, int height, int startAngle, int arcAngle)`*
 - *`void fillArc(int left, int top, int width, int height, int startAngle, int arcAngle)`*

Drawing Polygons

- It is possible to draw arbitrarily shaped figures using `drawPolygon()` and `fillPolygon()`, shown here:
 - *`void drawPolygon(int x[], int y[], int numPoints)`*
 - *`void fillPolygon(int x[], int y[], int numPoints)`*
- The polygon's endpoints are specified by the coordinate pairs contained within the x and y arrays.
- The number of points defined by these arrays is specified by numPoints.
- There are alternative forms of these methods in which the polygon is specified by a Polygon object.

Example:

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

public class shapeEx2 extends Canvas{
    public static void main(String[] args) {
        JFrame f= new JFrame("Drawing Shape");
        Canvas can= new shapeEx2();
        can.setSize(400,700);
        f.add(can);
        f.pack();
        f.setVisible(true);
        f.setDefaultCloseOperation(3);
    }
    public void paint(Graphics g)
    {
        // Draw lines.
        g.drawLine(0, 0, 100, 90);
        g.drawLine(0, 90, 100, 10);
        g.drawLine(40, 25, 250, 80);
```

```
// Draw rectangles.
```

```
g.drawRect(10, 150, 60, 50);
```

```
g.fillRect(100, 150, 60, 50);
```

```
g.drawRoundRect(190, 150, 60, 50, 15, 15);
```

```
g.fillRoundRect(280, 150, 60, 50, 30, 40);
```

```
// Draw Ellipses and Circles
```

```
g.drawOval(10, 250, 50, 50);
```

```
g.fillOval(90, 250, 75, 50);
```

```
g.drawOval(190, 260, 100, 40);
```

```
// Draw Arcs
```

```
g.drawArc(10, 350, 70, 70, 0, 180);
```

```
g.fillArc(60, 350, 70, 70, 0, 75);
```

```
// Draw a polygon
```

```
int xpoints[] = { 10, 200, 10, 200, 10};
```

```
int ypoints[] = { 450, 450, 650, 650, 450};
```

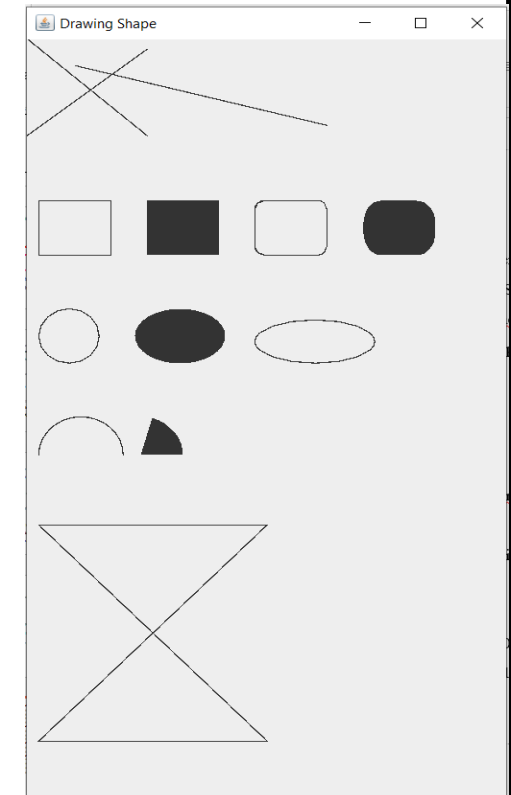
```
int num = 5;
```

```
g.drawPolygon(xpoints, ypoints, num);
```

```
}
```

```
}
```

Output



Working with Color

- Java supports color in a portable, device-independent fashion.
- The AWT color system allows you to specify any color you want.
- Color is encapsulated by the Color class.
- Color defines several constants (for example, Color.black) to specify a number of common colors.

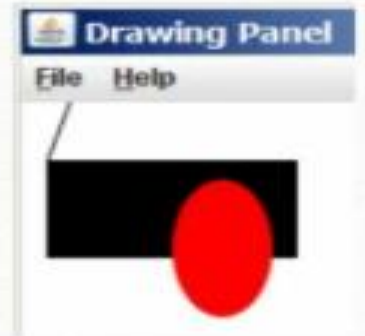
Color.CONSTANT_NAME

- Where CONSTANT_NAME is one of:
BLACK , BLUE, CYAN, DARK_GRAY, GREEN, RED, PINK, YELLOW, WHITE, ORANGE, MAGENTA etc.
- You can also create your own colors, using one of the color constructors.
 - ***Color(int red, int green, int blue)***
- Example:
 - **Color brown= new Color(192,128,64);**
- Where value of red, green and blue are ranges from 0 to 255

Using colors

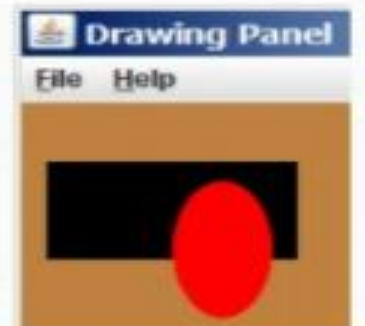
- Pass a `Color` to `Graphics` object's `setColor` method
 - Subsequent shapes will be drawn in the new color.

```
g.setColor(Color.BLACK);  
g.fillRect(10, 30, 100, 50);  
g.drawLine(20, 0, 10, 30);  
g.setColor(Color.RED);  
g.fillOval(60, 40, 40, 70);
```



- Pass a color to `DrawingPanel`'s `setBackground` method
 - The overall window background color will change.

```
Color brown = new Color(192, 128, 64);  
panel.setBackground(brown);
```



Working with Fonts

- The AWT supports multiple type fonts and provides flexibility by abstracting font-manipulation operations and allowing for dynamic selection of fonts.
- Fonts have a family name, a logical font name, and a face name.
- Fonts are encapsulated by the Font class

Creating and Selecting a Font

- To create a new font, construct a Font object that describes that font. One Font constructor has this general form:

Font(String fontName, int fontStyle, int pointSize);

- Here, fontName specifies the name of the desired font.
- All Java environments will support the following fonts: **Dialog**, **DialogInput**, **SansSerif**, **Serif**, and **Monospaced**.

- The style of the font is specified by `fontStyle`. It may consist of one or more of these three constants: **Font.PLAIN**, **Font.BOLD**, and **Font.ITALIC**.
- The size, in points, of the font is specified by `pointSize`
- To use a font that you have created, you must select it using **setFont()**, which is defined by `Component`. It has this general form:

void setFont(Font fontObj)

- Here, `fontObj` is the object that contains the desired font.

Example:

```
Font font= new Font("monospace",Font.BOLD|Font.ITALIC,20);  
g.setFont(font);  
g.drawString("Hello world", 20, 25);
```

Displaying image in swing

- For displaying image, we can use the method drawImage() of Graphics class.

boolean drawImage(Image imgObj, int left, int top, ImageObserver imgOb)

Or

boolean drawImage(Image imgObj, int left, int top, int width, int height ImageObserver imgOb)

- This displays the image passed in imgObj with its upper-left corner specified by left and top.
imgOb is a reference to a class that implements the ImageObserver interface

Example to display image

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

public class displayimage extends Canvas {
    public static void main(String[] args) {
        JFrame f1= new JFrame("Displaying Image");
        Canvas can= new displayimage();
        can.setSize(400,400);
        f1.add(can);
        f1.pack();
        f1.setVisible(true);
        f1.setDefaultCloseOperation(3);
    }
    public void paint(Graphics g)
    {
        try{
            BufferedImage img=ImageIO.read(getClass().getResource("lilis.jpg"));
            g.drawImage(img, 20,50,this);
        }catch(Exception e){ }
    }
}
```

Output

