

Chapter 5

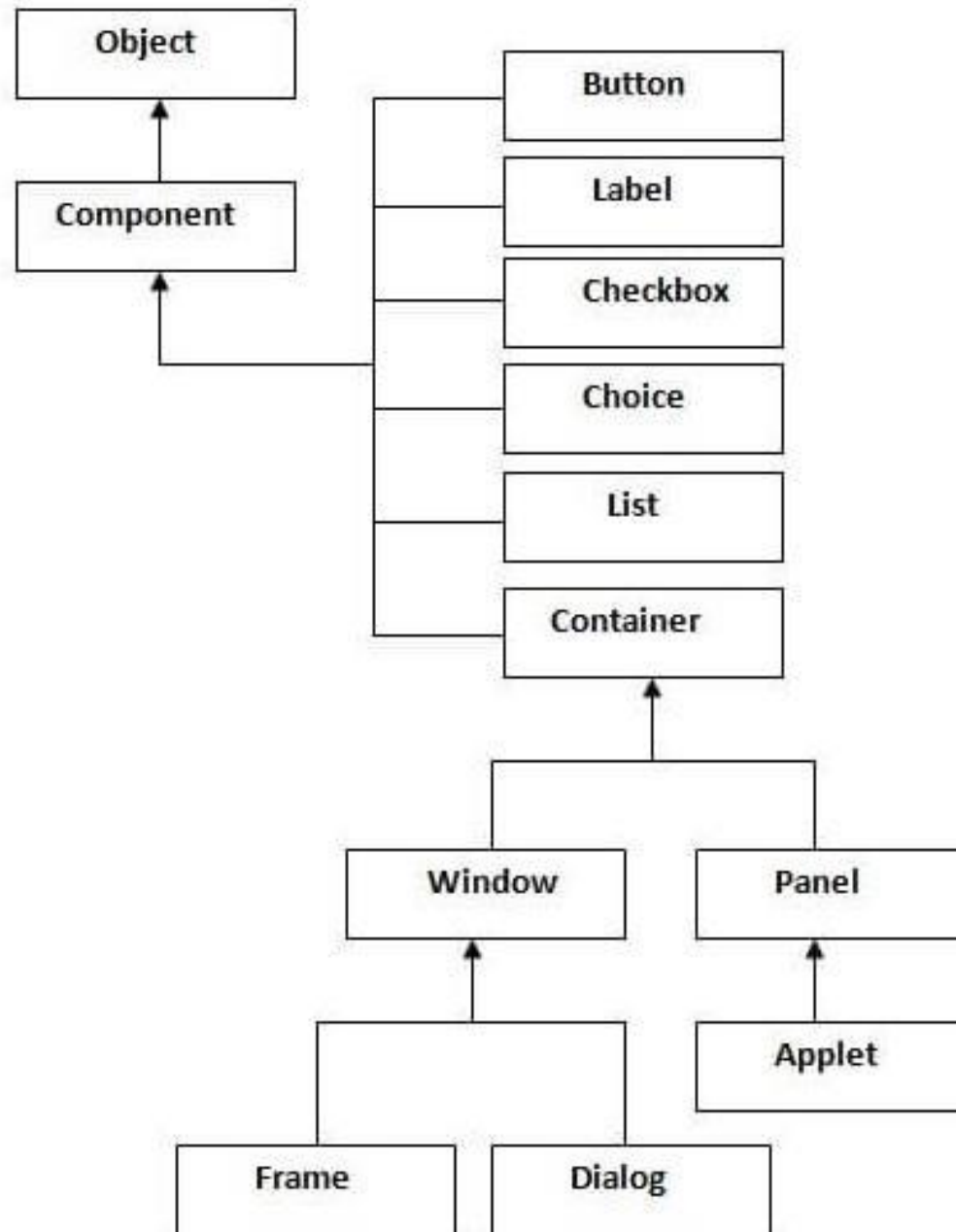
AWT and Swing

Components of AWT and Swing

- There are two main ways to create a graphical user interface (GUI) in Java.
- There is the AWT, which was the first one, and is older. Then there is Swing.
- Swing is a newer way of making a GUI.
- All objects in Swing are derived from AWT, and most objects in Swing start with the letter J.

- **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.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy



- ***Container***

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

- ***Window***

- The window is the container that has 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.

Method	Description
<code>public void add(Component c)</code>	inserts a component on this component.
<code>public void setSize(int width,int height)</code>	sets the size (width and height) of the component.
<code>public void setLayout(LayoutManager m)</code>	defines the layout manager for the component.
<code>public void setVisible(boolean status)</code>	changes the visibility of the component, by default false.

Java AWT Example

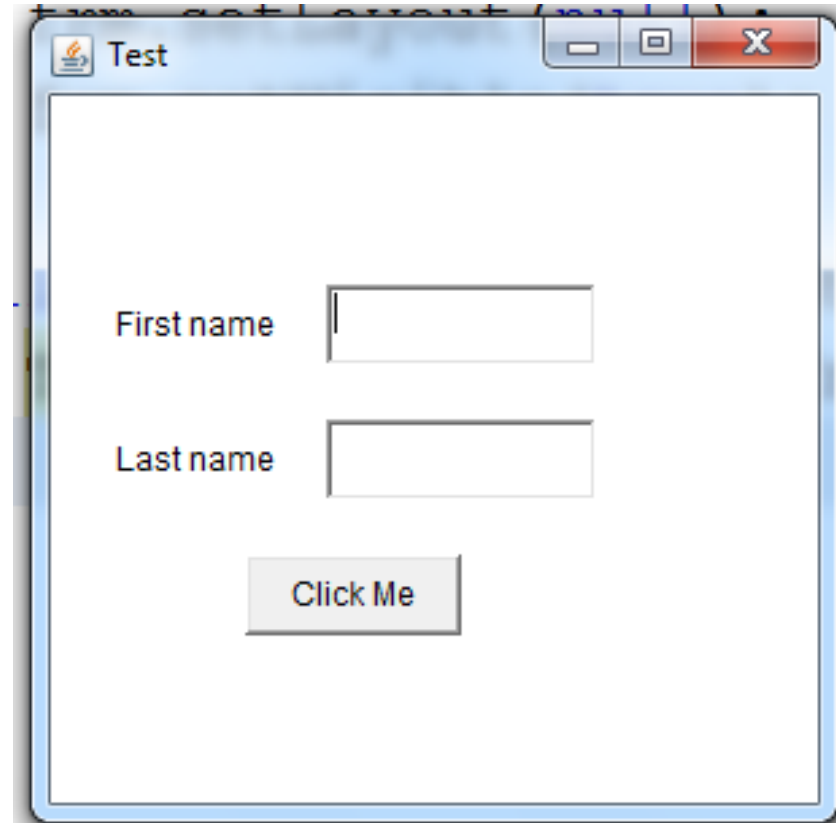
- To create simple awt example, you need a frame. There are two ways to create a frame in AWT.
 - By extending Frame class (inheritance)
 - By creating the object of Frame class (association)

AWT Example by Inheritance

```
import java.awt.*;  
public class TestFrame extends Frame {  
    public TestFrame(){  
        Label lblfname=new Label("First name");  
        Label lbllname=new Label("Last name");  
        TextField txtfname=new TextField("");  
        TextField txtlname=new TextField("");  
        Button b=new Button ("Click Me");
```

```
lblfname.setBounds(30, 100,80, 30);
txtfname.setBounds(110, 100,100, 30);
lbllname.setBounds(30, 150,80, 30);
txtlname.setBounds(110, 150, 100, 30);
b.setBounds(80, 200, 80, 30);
add(lblfname);
add(txtfname);
add(lbllname);
add(txtlname);
add(b);
setSize(300,300);
setLayout(null);
setVisible(true); }
public static void main(String args[]){
    TestFrame frm=new TestFrame();
}}
```

- The `setBounds(int xaxis, int yaxis, int width, int height)` method is used in the above example that sets the position of the awt button.



AWT Example by Association

```
import java.awt.*;  
public class TestFrame extends Frame {  
    public TestFrame(){  
        Frame frm=new Frame("Test");  
        Label lblfname=new Label("First name");  
        Label lbllname=new Label("Last name");  
        TextField txtfname=new TextField("");  
        TextField txtlname=new TextField("");  
        Button b=new Button ("Click Me");
```

```
lblfname.setBounds(30, 100,80, 30);
    txtfname.setBounds(110, 100,100, 30);
    lbllname.setBounds(30, 150,80, 30);
    txtlname.setBounds(110, 150, 100, 30);
    b.setBounds(80, 200, 80, 30);
    frm.add(lblfname);
    frm.add(txtfname);
    frm.add(lbllname);
    frm.add(txtlname);
    frm.add(b);
    frm.setSize(300,300);
    frm.setLayout(null);
    frm.setVisible(true);
}
public static void main(String args[]){
    TestFrame frm=new TestFrame();}}
```

Event and Listener (Java Event Handling)

- Changing the state of an object is known as an event.
 - For example, click on button, dragging mouse etc.
- The `java.awt.event` package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener

Steps to perform Event Handling

- Following steps are required to perform event handling:
 - Register the component with the Listener
 - Event handling

- ***Registration Methods***
- For registering the component with the Listener, many classes provide the registration methods. For example:
- **Button**
 - `public void addActionListener(ActionListener a){}`
- **MenuItem**
 - `public void addActionListener(ActionListener a){}`
- **TextField**
 - `public void addActionListener(ActionListener a){}`
 - `public void addTextListener(TextListener a){}`
- **TextArea**
 - `public void addTextListener(TextListener a){}`
- **Checkbox**
 - `public void addItemListener(ItemListener a){}`
- **Choice**
 - `public void addItemListener(ItemListener a){}`
- **List**
 - `public void addActionListener(ActionListener a){}`
 - `public void addItemListener(ItemListener a){}`

Java Event Handling Code

- We can put the event handling code into one of the following places:
 - Within class
 - Other class
 - Anonymous class

Java event handling by implementing ActionListener (within class)

```
import java.awt.*;  
import java.awt.event.*;  
class AEvent extends Frame implements ActionListener{  
    TextField tf;  
    AEvent(){  
  
        //create components  
        tf=new TextField();  
        tf.setBounds(60,50,170,20);  
        Button b=new Button("click me");  
        b.setBounds(100,120,80,30);  
        //register listener  
        b.addActionListener(this); //passing current instance
```

```
//add components and set size, layout and visibility  
add(b);add(tf);  
setSize(300,300);  
setLayout(null);  
setVisible(true);  
}  
public void actionPerformed(ActionEvent e){  
tf.setText("Welcome");  
}  
public static void main(String args[]){  
new AEvent();  
} }
```

Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
    TextField tf;
    AEvent2(){
        //create components
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
        b.setBounds(100,120,80,30);
        //register listener
        Outer o=new Outer(this);
        b.addActionListener(o);//passing outer class instance
        //add components and set size, layout and visibility
        add(b);add(tf);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String args[]){
        new AEvent2();
    }
}
```

//separate file Outer.java

import java.awt.event.*;

class Outer **implements** ActionListener{

AEvent2 obj;

Outer(AEvent2 obj){

this.obj=obj;

}

public void actionPerformed(ActionEvent e){

obj.tf.setText("welcome");

}

}

Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
    TextField tf;
    AEvent3(){
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
        b.setBounds(50,120,80,30);

        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                tf.setText("hello");
            }
        });
        add(b);add(tf);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String args[]){
        new AEvent3();
    }
}
```

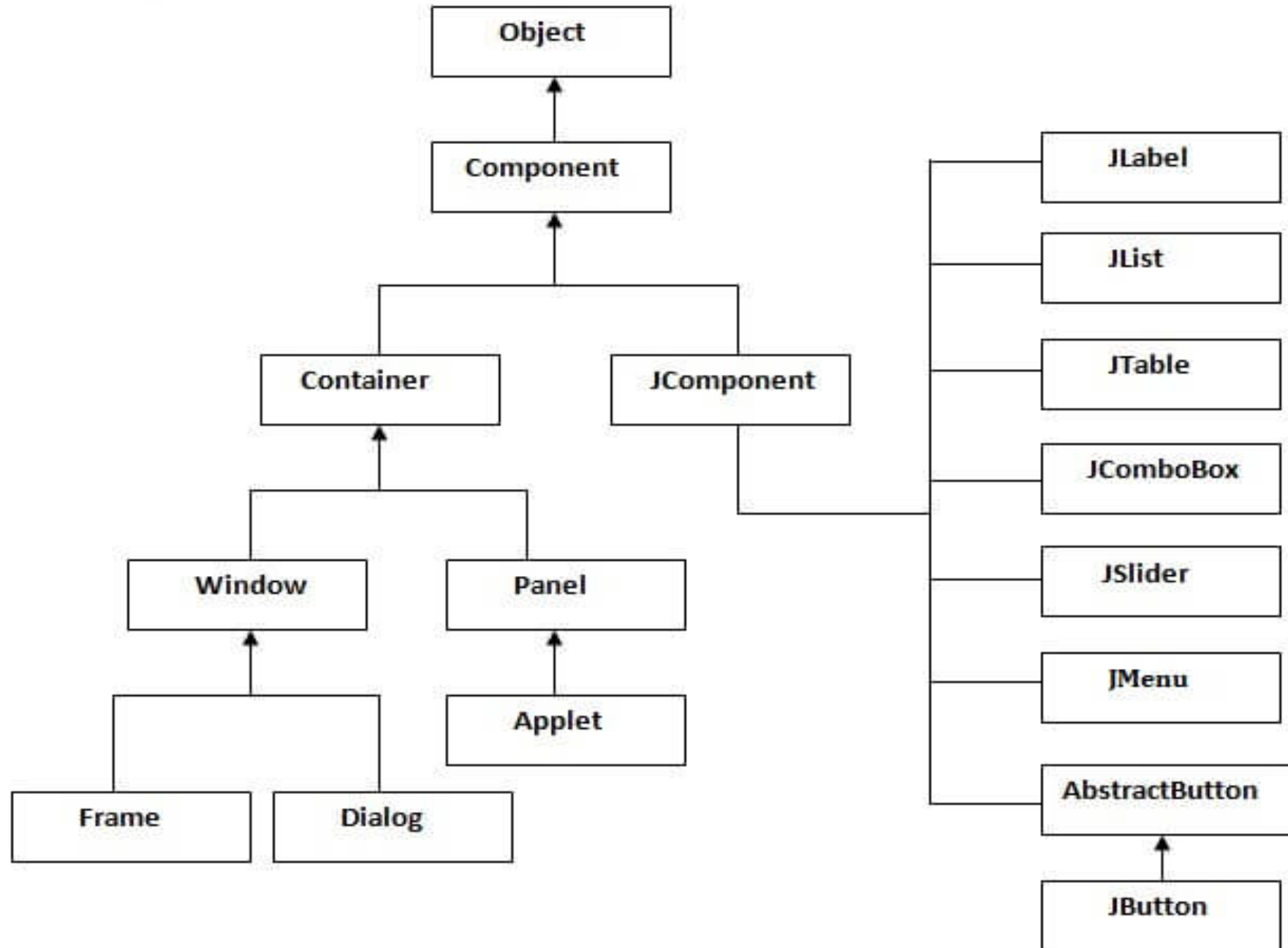
Java Swing

- Is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*.
- JFC are a set of GUI components which simplify the development of desktop 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 componentssuch 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



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.

- **Example**
- The following code finds the sum two integers by using message dialogs and input dialogs import javax.swing.*;

```
public class Addition {  
    public static void main(String args[]) {  
        String firstnumber=JOptionPane.showInputDialog("Enter  
the first number");  
        String  
secondnumber=JOptionPane.showInputDialog("Enter the  
second number");  
        int x,y,sum;  
        x=Integer.parseInt(firstnumber);  
        y=Integer.parseInt(secondnumber);  
        sum=x+y;  
        JOptionPane.showMessageDialog(null,"the sum of  
"+firstnumber+ "and  
"+secondnumber+"="+"sum","Addition",JOptionPane.PLAIN_  
MESSAGE);  
    }  
}
```

Layout Mangers

- **Layout managers** are provided to arrange GUI components in a container for presentation purposes.
- Programmers can use the layout managers for basic layout capabilities instead of determining the exact position and size of every GUI component.
- All layout managers implement the interface `LayoutManager` (in package `java.awt`).
- Class `Container`'s `setLayout` method takes an object that implements the `LayoutManager` interface as an argument.

- There are basically three ways for you to arrange components in a GUI:
- 1. Absolute positioning:** This provides the greatest level of control over a GUI's appearance.
 - ✓ By setting a Container's layout to null, you can specify the absolute position of each GUI component with respect to the upper-left corner of the Container.
 - ✓ If you do this, you also must specify each GUI component's size.
 - ✓ Programming a GUI with absolute positioning can be tedious, unless you have an integrated development environment (IDE) that can generate the code for you.

2. Layout managers: Using layout managers to position elements can be simpler and faster than creating a GUI with absolute positioning, but you lose some control over the size and the precise positioning of GUI components.

3. Visual programming in an IDE: IDEs provide tools that make it easy to create GUIs.

- ✓ Each IDE typically provides a GUI design tool that allows you to drag and drop GUI components from a tool box onto a design area.
- ✓ You can then position, size and align GUI components as you like.
- ✓ The IDE generates the Java code that creates the GUI.
- ✓ In addition, you can typically add event-handling code for a particular component by double-clicking the component.

- We have different layout managers. From these managers we will discuss three of them.

Layout manager	Description
FlowLayout	Default for <code>javax.swing.JPanel</code> . Places components sequentially (left to right) in the order they were added. It is also possible to specify the order of the components by using the <code>Container</code> method <code>add</code> , which takes a <code>Component</code> and an integer index position as arguments.
BorderLayout	Default for <code>JFrames</code> (and other windows). Arranges the components into five areas: <code>NORTH</code> , <code>SOUTH</code> , <code>EAST</code> , <code>WEST</code> and <code>CENTER</code> .
GridLayout	Arranges the components into rows and columns.

FlowLayout

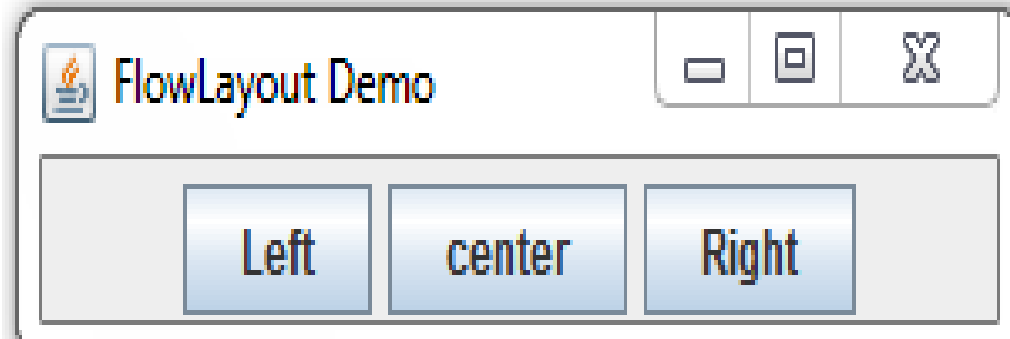
- FlowLayout is the simplest layout manager. GUI components are placed on a container from left to right in the order in which they are added to the container.
- When the edge of the container is reached, components continue to display on the next line.
- Class FlowLayout allows GUI components to be left aligned centered (the default) and right aligned.

Example

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FlowLayoutFrame extends JFrame implements
    ActionListener{
    JButton leftJButton,centerJButton,rightJButton;
    FlowLayout layout;
    Container container;
    public FlowLayoutFrame(){
        super("FlowLayout Demo");
        layout =new FlowLayout();
        container=getContentPane();
        setLayout(layout);
        leftJButton=new JButton("Left");
        add(leftJButton);
        leftJButton.addActionListener(this);
        centerJButton=new JButton("center");
```

```
add(centerJButton);
    centerJButton.addActionListener(this);
    rightJButton=new JButton("Right");
    add(rightJButton);
    rightJButton.addActionListener(this);
}
public void actionPerformed(ActionEvent e) {
    if(e.getSource()==leftJButton){
        layout.setAlignment(FlowLayout.LEFT);
        layout.layoutContainer(container);
    }
    if(e.getSource()==centerJButton){
        layout.setAlignment(FlowLayout.CENTER);
        layout.layoutContainer(container);
    }
}
```

```
if(e.getSource()==rightJButton){  
    layout.setAlignment(FlowLayout.RIGHT);  
    layout.layoutContainer(container);  
}  
}  
public static void main (String args[]){  
    FlowLayoutFrame flowlayoutframe=new FlowLayoutFrame();  
    flowlayoutframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    flowlayoutframe.setSize(300,70);  
    flowlayoutframe.setVisible(true);  
}
```



BorderLayout

- The BorderLayout layout manager (the default layout manager for a JFrame) arranges components into five regions: NORTH, SOUTH, EAST, WEST and CENTER. NORTH corresponds to the top of the container.
- A BorderLayout limits a Container to containing at most five components—one in each region.
- The component placed in each region can be a container to which other components are attached.

- The components placed in the NORTH and SOUTH regions extend horizontally to the sides of the container and are as tall as the components placed in those regions.
- The EAST and WEST regions expand vertically between the NORTH and SOUTH regions and are as wide as the components placed in those regions.
- The component placed in the CENTER region expands to fill all remaining space in the layout.

Example

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
public class BorderLayoutFrame extends JFrame implements  
    ActionListener{  
    JButton a=new JButton ("Hide North");  
    JButton b=new JButton ("Hide South");
```



```
JButton c=new JButton ("Hide West");
JButton d=new JButton ("Hide East");
JButton e=new JButton ("Center");
BorderLayout layout;
public BorderLayoutFrame(){
    super( "BorderLayout Demo" );
    layout = new BorderLayout( 5, 5 );
    setLayout(layout);
    a.addActionListener(this);
    b.addActionListener(this);
    c.addActionListener(this);
    d.addActionListener(this);
    e.addActionListener(this);
    add(a,BorderLayout.NORTH);
    add(b,BorderLayout.SOUTH);
    add(c,BorderLayout.EAST);
    add(d,BorderLayout.WEST);
    add(e,BorderLayout.CENTER);
}
```

```
public void actionPerformed(ActionEvent m) {  
    if(m.getSource()==a)  
        a.setVisible(false);  
    if(m.getSource()==b)  
        b.setVisible(false);  
    if(m.getSource()==c)  
        c.setVisible(false);  
    if(m.getSource()==d)  
        d.setVisible(false);  
    if(m.getSource()==e){  
        e.setVisible(false);  
    }  
}
```

```
if(a.isVisible()==false&&b.isVisible()==false&&c.isVisible()==false&&d.i  
sVisible()==false&&e.isVisible()==false)  
{  
    a.setVisible(true);  
    b.setVisible(true);  
    c.setVisible(true);  
    d.setVisible(true);  
    e.setVisible(true);  
}  
    layout.layoutContainer( getContentPane() );  
}  
public static void main( String args[] )  
{  
    BorderLayoutFrame borderLayoutFrame = new BorderLayoutFrame();  
    borderLayoutFrame.setDefaultCloseOperation(  
        JFrame.EXIT_ON_CLOSE );  
    borderLayoutFrame.setSize( 300, 200 ); // set frame size  
    borderLayoutFrame.setVisible( true ); // display frame  
} }
```

GridLayout

- The GridLayout layout manager divides the container into a grid so that components can be placed in rows and columns.
- Every Component in a GridLayout has the same width and height.
- Components are added to a GridLayout starting at the top-left cell of the grid and proceeding left to right until the row is full.
- Then the process continues left to right on the next row of the grid, and so on.

Example

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

public class GridLayoutFrame extends JFrame
    implements ActionListener{

    JButton a=new JButton ("One");
    JButton b=new JButton ("Two");
    JButton c=new JButton ("Three");
    JButton d=new JButton ("Four");
    JButton e=new JButton ("Five");
    JButton f=new JButton ("Six");
    boolean toggle=false ;
    Container container;
    GridLayout gridlayout1,gridlayout2;
```

```
public GridLayoutFrame(){
    super("GridLayout Demo");
    gridlayout1=new GridLayout(2,3,5,5);
    gridlayout2=new GridLayout(3,2);
    container=getContentPane();
    setLayout(gridlayout1);
    a.addActionListener(this);
    b.addActionListener(this);
    c.addActionListener(this);
    d.addActionListener(this);
    e.addActionListener(this);
    f.addActionListener(this);
    add(a);
    add(b);
    add(c);
    add(d);
    add(e);
    add(f);
}
```

@Override

```
public void actionPerformed(ActionEvent e) {  
    if(toggle)  
        container.setLayout(gridlayout1);  
    else  
        container.setLayout(gridlayout2);  
    toggle=!toggle;  
    container.validate();//validate recomputes the container's layout  
    based on the current layout manager for the Container and the  
    current set of displayed GUI components.  
}
```

```
public static void main( String args[] )  
{  
    GridLayoutFrame gridLayoutFrame = new GridLayoutFrame();  
    gridLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE  
        );  
    gridLayoutFrame.setSize( 300, 200 ); // set frame size  
    gridLayoutFrame.setVisible( true ); // display frame } }
```

Using Panels to Manage More Complex Layouts

- Complex GUIs (like in the example in this section) require that each component be placed in an exact location.
- They often consist of multiple panels, with each panel's components arranged in a specific layout.
- Class JPanel extends JComponent and JComponent extends class Container, so every JPanel is a Container.
- Thus, every JPanel may have components, including other panels, attached to it with Container method add.

Panel class

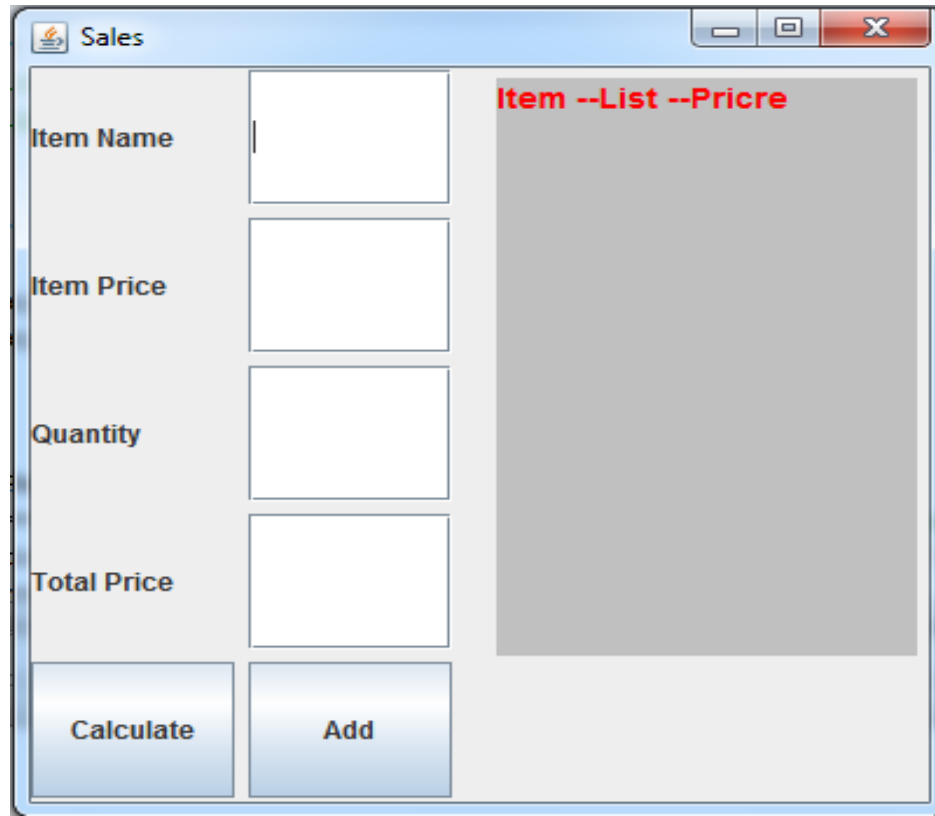
- The class Panel is the simplest container class.
- It provides space in which an application can attach any other component, including other panels.
- It uses FlowLayout as default layout manager.

Class constructors

S.N	Constructor & Description
1	Panel() Creates a new panel using the default layout manager.
2	Panel(LayoutManager layout) Creates a new panel with the specified layout manager.

Example:

- the following example is for calculating price for items. The figure below shows the GUI of the program.



The image shows a Java Swing window titled "Sales" with a standard Mac OS X-style title bar (minimize, maximize, close buttons). The window contains a form with four input fields on the left, each with a label to its left: "Item Name", "Item Price", "Quantity", and "Total Price". Below these fields are two buttons: "Calculate" and "Add". To the right of the input fields is a large, empty rectangular area with a light gray background, intended for displaying a list of items. At the top of this area, the text "Item --List --Pricre" is written in red, likely representing a header for a table or list.

- To create the following form, we have created two panels, p1 and p2.
- P1 have gridlayout with five rows and 2 columns whereas p2 is set to have flowlayout.
- The textarea is added to p2 panel and the rest of the components are on p1.
- Our frame has borderlayout manager therefore that is why we have p1 in west and p2 east side of the frame.
- The following is the code for the layout manager with action listeners set to calculate the prices

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class CalcPrice extends JFrame implements ActionListener{
    JPanel p1,p2;
    JLabel itemname,itemprice,qty,totalprice;
    JButton calc,add;
    JTextField txtitemname,txtitemprice,txtqty,txttotalprice;
    JTextArea report;
    double total=0,sum=0;
    public CalcPrice(){
        super("Sales");
        p1=new JPanel(new GridLayout(5,2,6,6));
        p2=new JPanel(new FlowLayout());
        itemname=new JLabel("Item Name");
        itemprice=new JLabel("Item Price");
        qty=new JLabel("Quantity ");
        totalprice=new JLabel("Total Price");
```

```
calc=new JButton("Calculate");
    calc.addActionListener(this);
    add=new JButton("Add");
    add.addActionListener(this);
    txtitemname=new JTextField("");
    txtitemprice=new JTextField("");
    txtqty=new JTextField("");
    txttotalprice=new JTextField("");
    report=new JTextArea("Item --List --Pricre \n",15,15);
    report.setEditable(false);
    report.setBackground(Color.lightGray);
    report.setForeground(Color.red);
    report.setFont(new Font( "SansSerif", Font.BOLD, 14));
    p1.add(itemname);
    p1.add(txtitemname);
    p1.add(itemprice);
    p1.add(txtitemprice);
    p1.add(qty);
    p1.add(txtqty);
    p1.add(totalprice);
    p1.add(txttotalprice);
    p1.add(calc);
    p1.add(add);
    p2.add(report);
    add(p1,BorderLayout.WEST);
    add(p2,BorderLayout.EAST);
}
```

```

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource()==add &&
        txtitemname.getText().equals("")&txtitemprice.getText().equals("")&&txtqty.getText().equals(""))
        JOptionPane.showMessageDialog(null,"Please insert
        data","Error",JOptionPane.WARNING_MESSAGE);
    else if(e.getSource()==add){
        sum=Double.parseDouble(txtqty.getText())*Double.parseDouble(txtitemprice.getText());
        total=total+sum;
        report.append(txtitemname.getText()+"---"+txtqty.getText()+"X"+txtitemprice.getText()+"---
        "+sum+"\n");
        txtitemname.setText("");
        txtitemprice.setText("");
        txtqty.setText("");
    }
    else if (e.getSource()==calc){
        report.append("Total-----"+total);
        txttotalprice.setText(""+total);
    }
    else
        JOptionPane.showMessageDialog(null,"invalid operation");
}

public static void main(String args[]){
    CalcPrice calobj=new CalcPrice();
    calobj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    calobj.setSize(400,400);
    calobj.setVisible(true);
}
}

```