AWT

The Java programming language class library provides a user interface toolkit called the Abstract Windowing Toolkit, or the AWT. The AWT is both powerful and flexible.

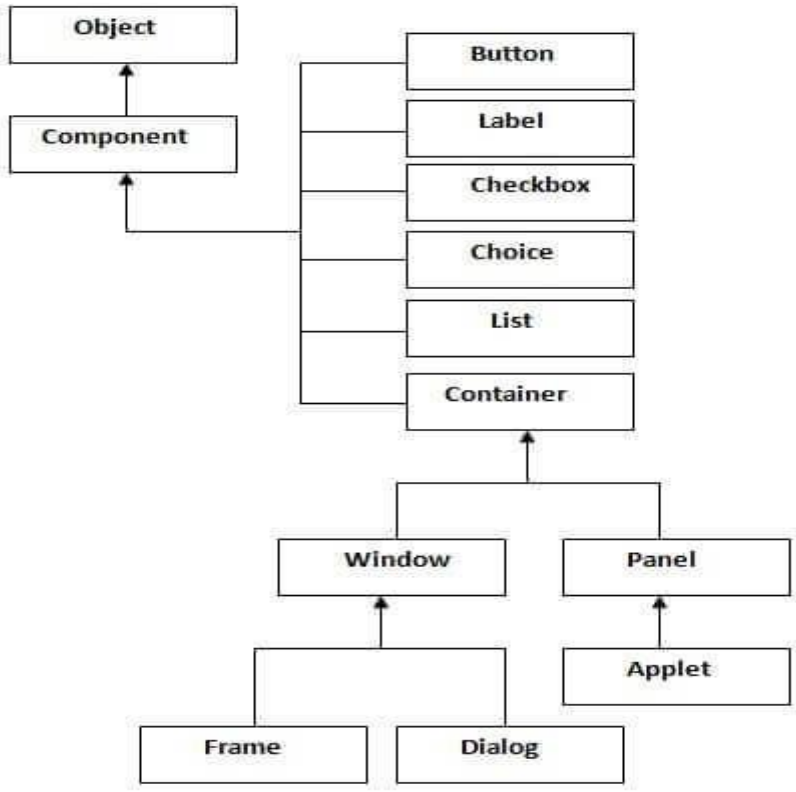
The basic idea behind the AWT is that a graphical Java program is a set of nested components, starting from the outermost window all the way down to the smallest UI component. Components can include things we can actually see on the screen, such as windows, menu bars, buttons, and text fields, and they can also include containers, which in turn can contain other components.

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.

The hierarchy of Java AWT classes is given below:



Container

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

- **Window**
  The window is the container that has no borders and menu bars. We 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 AWT Example

To create simple awt example, we 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

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame
{
        First()
        {
                Button b=new Button("click me");
                b.setBounds(30,100,80,30);          // setting button position
                add(b);                             //adding button into frame
                setSize(300,300);                   //frame size 300 width and 300 height
```

```
        setLayout(null);                    //no layout manager
        setVisible(true);       //now frame will be visible, by default not visible
    }
    public static void main(String args[])
    {
        First f=new First();
    }
}
```

OUTPUT

AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First2
{
    First2()
    {
        Frame f=new Frame();
        Button b=new Button("click me");
```

```
            b.setBounds(30,50,80,30);
            f.add(b);
            f.setSize(300,300);
            f.setLayout(null);
            f.setVisible(true);
        }
        public static void main(String args[])
        {
            First2 f=new First2();
        }
}
```

OUTPUT

The Basic User Interface Components

The simplest form of awt component is the basic UI component. Because an applet is a container, we can put other awt components-such as UI components or other containers-into it.

The basic UI components are labels, buttons, check boxes, choice menus, and text fields. To add a component to a panel (such as our applet, for example), use the `add()` method:

```
public void init() {
    Button b = new Button("OK");
    add(b);
}
```

Here the `add()` method refers to the current applet.

Labels

The simplest form of UI component is the label, which is, effectively, a text string that we can use to label other UI components. Labels are not editable; they just label other components on the screen.

A *label* is an un-editable text string that acts as a description for other awt components.

To create a label, use one of the following constructors:

- `Label()` creates an empty label, with its text aligned left.
- `Label(`*`String`*`)` creates a label with the given text string, also aligned left.
- `Label(`*`String`*`, `*`int`*`)` creates a label with the given text string and the given alignment. The available alignment numbers are stored in class variables in `Label`, making them easier to remember: `Label.RIGHT`, `Label.LEFT`, and `Label.CENTER`.

```
import java.awt.*;
public class LabelTest extends java.applet.Applet {
  public void init() {
    setFont(new Font ("Helvetica", Font.BOLD, 14));
    setLayout(new GridLayout(3,1));
    add(new Label("aligned left", Label.LEFT));
    add(new Label("aligned center", Label.CENTER));
    add(new Label("aligned right", Label.RIGHT));
  }
}
```

When we have a `Label` object, we can use methods defined in the `Label` class to get and set the values of the text, as shown in Table:

TableLabel methods.

Method	Action
`getText()`	Returns a string containing this label's text
`setText(String)`	Changes the text of this label

`getAlignment()`	Returns an integer representing the alignment of this label:
	`0` is `Label.LEFT`
	`1` is `Label.CENTER`
	`2` is `Label.RIGHT`

`setAlignment(int)`	Changes the alignment of this label to the given integer-use the class variables listed in the `getAlignment()` method

Buttons

The second user interface component to explore is the button. Buttons are simple UI components that trigger some action in our interface when they are pressed.

To create a button, use one of the following constructors:

- `Button()` creates an empty button with no label.
- `Button(String)` creates a button with the given string as a label.

```
public class ButtonTest extends java.applet.Applet {
  public void init() {
    add(new Button("Rewind"));
    add(new Button("Play"));
    add(new Button("Fast Forward"));
    add(new Button("Stop"));
  }
}
```

Check Boxes

Check boxes are user-interface components that have two states: on and off (or checked and unchecked, selected and unselected, true and false, and so on). Unlike buttons, check boxes usually don't trigger direct actions in a UI, but instead are used to indicate optional features of some other action.

Check boxes can be used in two ways:

- Nonexclusive: Given a series of check boxes, any of them can be selected.

- Exclusive: Given a series, only one check box can be selected at a time.

Nonexclusive check boxes can be created by using the `Checkbox` class. We can create a check box using one of the following constructors:

- `Checkbox()` creates an empty check box, unselected.
- `Checkbox(String)` creates a check box with the given string as a label.
- `Checkbox(String, null, boolean)` creates a check box that is either selected or deselected based on whether the boolean argument is `true` or `false`, respectively. (The `null` is used as a placeholder for a group argument. Only radio buttons have groups, as you'll learn in the next section.)

Five check boxes, one selected.

```
import java.awt.*;

public class CheckboxTest extends java.applet.Applet {

  public void init() {
    setLayout(new FlowLayout(FlowLayout.LEFT));
    add(new Checkbox("Shoes"));
    add(new Checkbox("Socks"));
    add(new Checkbox("Pants"));
    add(new Checkbox("Underwear", null, true));
    add(new Checkbox("Shirt"));
  }

}
```

Table lists some of the check box methods.

Table Check box methods.

Method	Action
`getLabel()`	Returns a string containing this check box's label
`setLabel(String)`	Changes the text of the check box's label
`getState()`	Returns `true` or `false`, based on whether the check box is selected
`setState(boolean)`	Changes the check box's state to selected (`true`) or unselected (`false`)

Radio Buttons

Radio buttons have the same appearance as check boxes, but only one in a series can be selected at a time. To create a series of radio buttons, first create an instance of `CheckboxGroup`:

```
CheckboxGroup cbg = new CheckboxGroup();
```

Then create and add the individual check boxes using the constructor with three arguments (the first is the label, the second is the group, and the third is whether that check box is selected). Note that because radio buttons, by definition, have only one in the group selected at a time, the last `true` to be added will be the one selected by default:

```
add(new Checkbox("Yes", cbg, true);
add(new Checkbox("No", cbg, false);
```

Here's a simple example:

Six radio buttons (exclusive check boxes), one selected.

```
import java.awt.*;

public class CheckboxGroupTest extends java.applet.Applet {

  public void init() {
    setLayout(new FlowLayout(FlowLayout.LEFT));
    CheckboxGroup cbg = new CheckboxGroup();

    add(new Checkbox("Red", cbg, false));
    add(new Checkbox("Blue", cbg, false));
    add(new Checkbox("Yellow", cbg, false));
    add(new Checkbox("Green", cbg, true));
    add(new Checkbox("Orange", cbg, false));
    add(new Checkbox("Purple", cbg, false));
  }

}
```

Choice Menus

The choice menu is a more complex UI component than labels, buttons, or check boxes. Choice menus are pop-up (or pull-down) menus from which we can select an item. The menu then displays that choice on the screen. The function of a choice menu is the same across platforms, but its actual appearance may vary from platform to platform.

Choice menus are pop-up menus of items from which we can choose one item. To create a choice menu, create an instance of the `Choice` class and then use the `addItem()` method to add individual items to it in the order in which they should appear.

Finally, add the entire choice menu to the panel in the usual way. Here's a simple program that builds a choice menu of fruits;

A choice menu.

```
import java.awt.*;

public class ChoiceTest extends java.applet.Applet {

  public void init() {
    Choice c = new Choice();

    c.addItem("Apples");
    c.addItem("Oranges");
    c.addItem("Strawberries");
    c.addItem("Blueberries");
    c.addItem("Bananas");

    add(c);
  }
}
```

Table Choice menu methods.

Method	Action
`getItem(int)`	Returns the string item at the given position (items inside a choice begin at `0`, just like arrays)
`countItems()`	Returns the number of items in the menu
`getSelectedIndex()`	Returns the index position of the item that's selected
`getSelectedItem()`	Returns the currently selected item as a string
`select(int)`	Selects the item at the given position
`select(String)`	Selects the item with the given string

Text Fields

The text fields allow us to enter and edit text. Text fields are generally only a single line and do not have scrollbars.

Text fields are different from labels in that they can be edited; labels are good for just displaying text, text fields for getting text input from the user.

Text fields provide an area where we can enter and edit a single line of text.

To create a text field, use one of the following constructors:

- `TextField()` creates an empty `TextField` that is 0 characters wide (it will be resized by the current layout manager).
- `TextField(int)` creates an empty text field. The integer argument indicates the minimum number of characters to display.
- `TextField(String)` creates a text field initialized with the given string. The field will be automatically resized by the current layout manager.

- `TextField(String, int)` creates a text field some number of characters wide (the integer argument) containing the given string. If the string is longer than the width, you can select and drag portions of the text within the field, and the box will scroll left or right.

For example, the following line creates a text field 30 characters wide with the string `"Enter Your Name"` as its initial contents:

```
TextField tf = new TextField("Enter Your Name", 30);
add(tf);
```

Three text fields to allow input from the user.

```
add(new Label("Enter your Name"));
add(new TextField("your name here", 45));
add(new Label("Enter your phone number"));
add(new TextField(12));
add(new Label("Enter your password"));
TextField t = new TextField(20);
t.setEchoCharacter('*');
add(t);
```

Text field methods.

Method	Action
`getText()`	Returns the text this text field contains (as a string)
`setText(String)`	Puts the given text string into the field
`getColumns()`	Returns the width of this text field
`select(int, int)`	Selects the text between the two integer positions (positions start from `0`)
`selectAll()`	Selects all the text in the field
`isEditable()`	Returns `true` or `false` based on whether the text is editable
`setEditable(boolean)`	`true` (the default) enables text to be edited; `false` freezes the text
`getEchoChar()`	Returns the character used for masking input
`echoCharIsSet()`	Returns `true` or `false` based on whether the field has a masking character

Container

Containers contain and control the layout of components. Containers are themselves components, and can thus be placed inside other containers. In the AWT, all containers are instances of class Container or one of its subtypes.

The AWT provides four container classes. They are class Window and its two subtypes -- class Frame and class Dialog -- as well as the Panel class. In addition to the containers provided by the AWT, the Applet class is a container -- it is a subtype of the Panel class and can therefore hold components.

Window	A top-level display surface (a window). An instance of the Window class is not attached to nor embedded within another container. An instance of the Window class has no border and no title.
Frame	A top-level display surface (a window) with a border and title. An instance of the Frame class may have a menu bar. It is otherwise very much like an instance of the Window class.
Dialog	A top-level display surface (a window) with a border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class.
Panel	A generic container for holding components. An instance of the Panel class provides a container to which to add components.

Creating a container

Before adding the components that make up a user interface, the programmer must create a container. When building an application, the programmer must first create an instance of class Window or class Frame. When building an applet, a frame (the browser window) already exists. Since the Applet class is a subtype of the Panel class, the programmer can add the components to the instance of the Applet class itself.

The code given below creates an empty frame. The title of the frame ("Example 1") is set in the call to the constructor. A frame is initially invisible and must be made visible by invoking its show() method.

```
import java.awt.*;
public class Example1
{
  public static void main(String [] args)
  {
    Frame f = new Frame("Example 1");
    f.show();
  }
}
```

The code for deriving the new class from class Applet is shown below.

```
import java.awt.*;
public class Example1b extends java.applet.Applet
{
  public static void main(String [] args)
  {
    Frame f = new Frame("Example 1b");
    Example1b ex = new Example1b();
    f.add("Center", ex);
    f.pack();
    f.show();
  }
}
```

A frame with an empty applet

Adding components to a container

Components are added to containers via a container's add() method. There are three basic forms of the add() method. The method to use depends on the container's layout manager.

The code given below adds the creation of two buttons into the Applet. The creation is performed in the init() method because it is automatically called during applet initialization. Therefore, no matter how the program is started, the buttons are created, because init() is called by either the browser or by the main() method.

```
import java.awt.*;
public class Example3 extends java.applet.Applet
{
  public void init()
  {
    add(new Button("One"));
    add(new Button("Two"));
  }
  public Dimension preferredSize()
  {
    return new Dimension(200, 100);
  }
  public static void main(String [] args)
  {
    Frame f = new Frame("Example 3");
    Example3 ex = new Example3();
    ex.init();
```

```
    f.add("Center", ex);
    f.pack();
    f.show();
  }
}
```

Swing- based GUI

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

There are many differences between java awt and swing that are given below.

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**.

Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

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

Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
import javax.swing.*;
public class FirstSwingExample
{
        public static void main(String[] args)
        {
                JFrame f=new JFrame();              //creating instance of JFrame
                JButton b=new JButton("click");     //creating instance of JButton
                b.setBounds(130,100,100, 40);           //x axis, y axis, width, height
                 f.add(b);                                  //adding button in JFrame
```

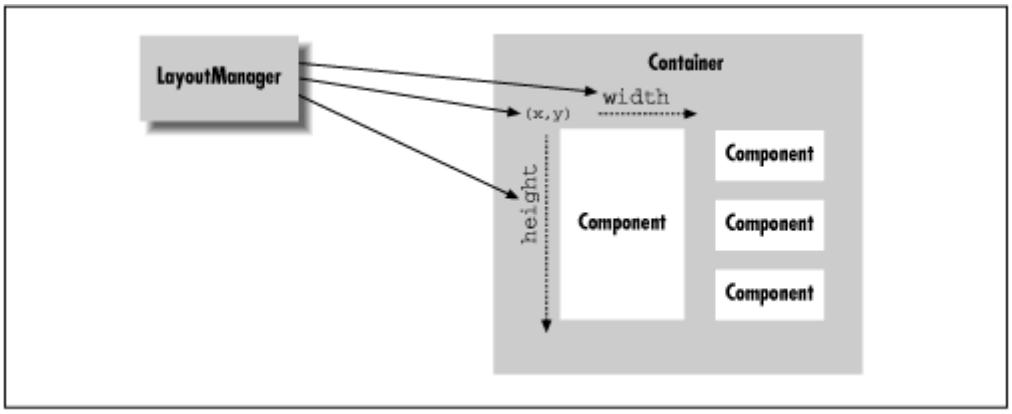
```
        f.setSize(400,500);                    //400 width and 500 height
        f.setLayout(null);                     //using no layout managers
        f.setVisible(true);                    //making the frame visible
    }
}
```

Layout Manager

The Layout managers enable us to control the way in which visual components are arranged in the GUI forms by determining the size and position of components within the containers.

A layout manager arranges the child components of a container, as shown in below figure. It positions and sets the size of components within the container's display area according to a particular layout scheme.

Figure : LayoutManager at work



Every container has a default layout manager; therefore, when we make a new container, it comes with a `LayoutManager` object of the appropriate type. Below, we set the layout manager of a container to a `BorderLayout`:

```
setLayout ( new BorderLayout() );
```

Types of LayoutManager

There are 6 layout managers in Java

- **FlowLayout**: It arranges the components in a container like the words on a page. It fills the top line from **left to right and top to bottom**. The components are arranged in the order as they are added i.e. first components appears at top left, if the container is not wide enough to display all the components, it is wrapped around the line. Vertical and horizontal gap between components can be controlled. The components can be **left, center or right aligned.**

- **BorderLayout**: It arranges all the components along the edges or the middle of the container i.e. **top, bottom, right and left** edges of the area. The components added to the top or bottom gets its preferred height, but its width will be the width of the container and also the components added to the left or right gets its preferred width, but its height will be the remaining height of the container. The components added to the center gets neither its preferred height or width. It covers the remaining area of the container.

- **GridLayout**: It arranges all the components in a grid of **equally sized cells**, adding them from the **left to right** and **top to bottom**. Only one component can be placed in a cell and each region of the grid will have the same size. When the container is resized, all cells are automatically resized. The order of placing the components in a cell is determined as they were added.
- **GridBagLayout**: It is a powerful layout which arranges all the components in a grid of cells and maintains the aspect ration of the object whenever the container is resized. In this layout, cells may be different in size. It assigns a consistent horizontal

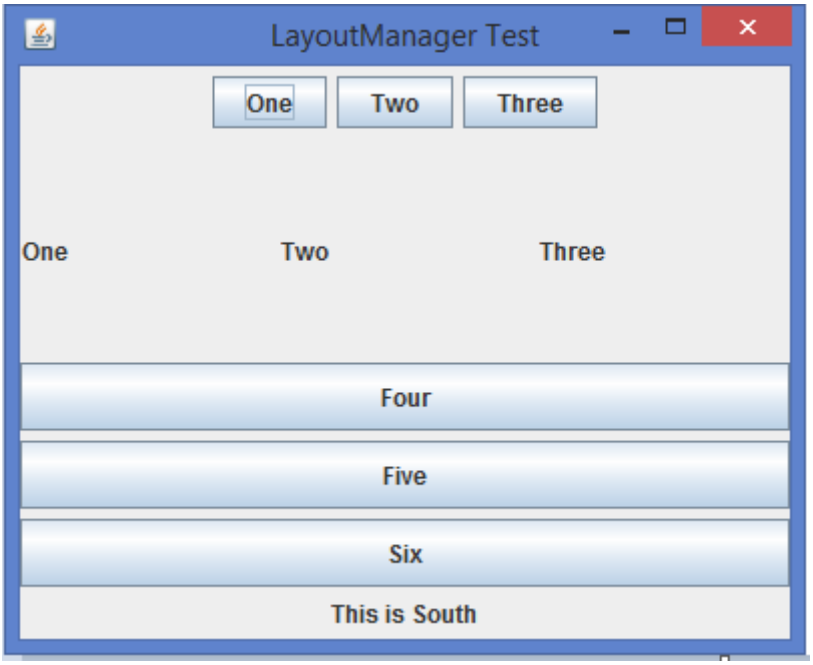
and vertical gap among components. It allows us to specify a default alignment for components within the columns or rows.

- **BoxLayout**: It arranges multiple components in either **vertically or horizontally**, but not both. The components are arranged from **left to right or top to bottom**. If the components are aligned **horizontally**, the height of all components will be the same and equal to the largest sized components. If the components are aligned **vertically**, the width of all components will be the same and equal to the largest width components.
- **CardLayout**: It arranges two or more components having the same size. The components are **arranged in a deck**, where all the cards of the same size and the **only top card are visible at any time**. The first component added in the container will be kept at the top of the deck. The default gap at the left, right, top and bottom edges are zero and the card components are displayed either **horizontally or vertically.**

Example

```
import java.awt.*;
import javax.swing.*;
public class LayoutManagerTest extends JFrame {
    JPanel flowLayoutPanel1, flowLayoutPanel2, gridLayoutPanel1, gridLayoutPanel2, gridLayoutPanel3;
  JButton one, two, three, four, five, six;
  JLabel bottom, lbl1, lbl2, lbl3;
  public LayoutManagerTest() {
    setTitle("LayoutManager Test");
    setLayout(new BorderLayout()); // Set BorderLayout for JFrame
    flowLayoutPanel1 = new JPanel();
    one = new JButton("One");
    two = new JButton("Two");
    three = new JButton("Three");
          flowLayoutPanel1.setLayout(new FlowLayout(FlowLayout.CENTER)); // Set FlowLayout Manager
    flowLayoutPanel1.add(one);
    flowLayoutPanel1.add(two);
    flowLayoutPanel1.add(three);
    flowLayoutPanel2 = new JPanel();
    bottom = new JLabel("This is South");
        flowLayoutPanel2.setLayout (new FlowLayout(FlowLayout.CENTER)); // Set FlowLayout Manager
    flowLayoutPanel2.add(bottom);
    gridLayoutPanel1 = new JPanel();
    gridLayoutPanel2 = new JPanel();
    gridLayoutPanel3 = new JPanel();
    lbl1 = new JLabel("One");
```

```
    lbl2 = new JLabel("Two");
    lbl3 = new JLabel("Three");
    four = new JButton("Four");
    five = new JButton("Five");
    six = new JButton("Six");
    gridLayoutPanel2.setLayout(new GridLayout(1, 3, 5, 5)); // Set GridLayout Manager
    gridLayoutPanel2.add(lbl1);
    gridLayoutPanel2.add(lbl2);
    gridLayoutPanel2.add(lbl3);
    gridLayoutPanel3.setLayout(new GridLayout(3, 1, 5, 5)); // Set GridLayout Manager
    gridLayoutPanel3.add(four);
    gridLayoutPanel3.add(five);
    gridLayoutPanel3.add(six);
    gridLayoutPanel1.setLayout(new GridLayout(2, 1)); // Set GridLayout Manager
    gridLayoutPanel1.add(gridLayoutPanel2);
    gridLayoutPanel1.add(gridLayoutPanel3);
    add(flowLayoutPanel1, BorderLayout.NORTH);
    add(flowLayoutPanel2, BorderLayout.SOUTH);
    add(gridLayoutPanel1, BorderLayout.CENTER);
    setSize(400, 325);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setVisible(true);
  }
  public static void main(String args[]) {
    new LayoutManagerTest();
  }
}
```


LayoutManager Test
One
Two
Three
One
Two
Three
Four
Five
Six
This is South