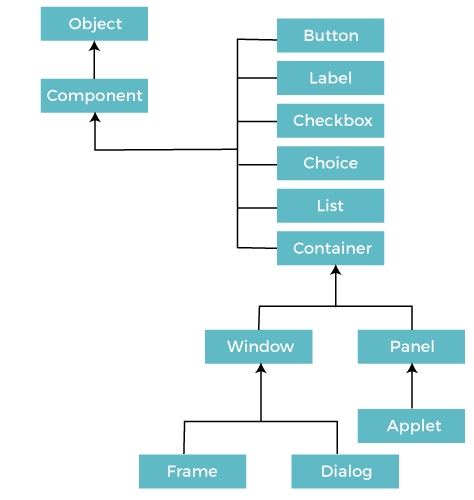
**Java AWT** (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The java.awt [package](https://www.javatpoint.com/package) provides [classes](https://www.javatpoint.com/object-and-class-in-java) for AWT API such as [TextField](https://www.javatpoint.com/java-awt-textfield), [Label](https://www.javatpoint.com/java-awt-label), [TextArea](https://www.javatpoint.com/java-awt-textarea), RadioButton, [CheckBox](https://www.javatpoint.com/java-awt-checkbox), [Choice](https://www.javatpoint.com/java-awt-choice), [List](https://www.javatpoint.com/java-awt-list) etc.

ava AWT Hierarchy

The hierarchy of Java AWT classes are given below.



**Components**

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

**Container**

The Container is a component in AWT that can contain another components like [buttons](https://www.javatpoint.com/java-awt-button), textfields, labels etc. The classes that extends Container class are known as container such as **Frame, Dialog** and **Panel**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

#### **Note: A container itself is a component (see the above diagram), therefore we can add a container inside container.**

**Types of containers:**

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

**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. We need to create an instance of Window class to create this container.

**Panel**

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

**Frame**

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

**Useful Methods of Component Class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void add(Component c) | Inserts a component on this component. |
| public void setSize(int width,int height) | Sets the size (width and height) of the component. |
| public void setLayout(LayoutManager m) | Defines the layout manager for the component. |
| public void setVisible(boolean status) | 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 GUI using Frame in AWT.

1. By extending Frame class (**inheritance**)
2. 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.

**AWTExample1.java**

// importing Java AWT class

**import** java.awt.\*;

// extending Frame class to our class AWTExample1

**public** **class** AWTExample1 **extends** Frame{

     // initializing using constructor

 AWTExample1()

{

      // creating a button

      Button b = **new** Button("Click Me!!");

      // setting button position on screen

      b.setBounds(30,100,80,30);

      // adding button into frame

      add(b);

      // frame size 300 width and 300 height

      setSize(300,300);

      // setting the title of Frame

      setTitle("This is our basic AWT example");

      // no layout manager

      setLayout(**null**);

      // now frame will be visible, by default it is not visible

      setVisible(**true**);

}

// main method

**public** **static** **void** main(String args[])

{

// creating instance of Frame class

AWTExample1 f = **new** AWTExample1();

  }

}

### AWT Example by Association

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

**AWTExample2.java**

// importing Java AWT class

**import** java.awt.\*;

// class AWTExample2 directly creates instance of Frame class

**class** AWTExample2 {

   // initializing using constructor

    AWTExample2() {

       // creating a Frame

       Frame f = **new** Frame();

       // creating a Label

       Label l = **new** Label("Employee id:");

       // creating a Button

       Button b = **new** Button("Submit");

       // creating a TextField

       TextField t = **new** TextField();

       // setting position of above components in the frame

       l.setBounds(20, 80, 80, 30);

       t.setBounds(20, 100, 80, 30);

       b.setBounds(100, 100, 80, 30);

       // adding components into frame

       f.add(b);

       f.add(l);        f.add(t);

      // frame size 300 width and 300 height

      f.setSize(400,300);

       // setting the title of frame

       f.setTitle("Employee info");

       // no layout

       f.setLayout(**null**);

       // setting visibility of frame

       f.setVisible(**true**);

}

// main method

**public** **static** **void** main(String args[]) {

// creating instance of Frame class

AWTExample2 awt\_obj = **new** AWTExample2();

}

}

**Basic AWT Controls**

**Container Class**

It represents objects in graphical representation and it is an abstract class in the GUI interface.

The following are the list of commonly used UI elements in the GUI or commonly [known as Graphical User Interface](https://www.educba.com/what-is-gui/).

**1. Label**

A label is a user for placing text inside the container. A label is used only for inputting text. The label does not imply that the text can be altered or it can be used as a button which can be further worked upon.

**Syntax:**

Label n=new Label("Name:",Label.CENTER);

**2. Button**

This command generates a button in the User Interface. Clicking on the button would move the command to another page or another web server which is used to show several other outputs in the user interface page.

**Syntax:**

a1=new Button("submit");

a2=new Button("cancel");

**3. Checkbox**

There can be a certain question and the checkbox is used to determine the true or false nature of the question being asked. If the checkbox is ticked then it means that the said question is true which if it is unchecked it means that the said question is false. It is basically a true or false [state in Java programming language](https://www.educba.com/java-programming-language-features/).

**Syntax:**

Checkbox checkbox1 = new Checkbox("Hello World");

**4. Checkbox Group**

As the name implies the checkbox group is a set of checkboxes that are being used in the programming language. There are many checkboxes that are being used and hence the group of checkboxes is known as the checkbox group.

**Syntax:**

CheckboxGroup cb = new CheckboxGroup();

Checkbox checkBox1 = new Checkbox("Hello", cb, true);

checkBox1.setBounds (100,100, 50,50);

**5. List**

The list gives a scrolling list of items for the user. The scrolling list of items is also being set by the user. The user sets the scrolling list of items such as Fruits, Vegetables, some questionnaire or other facts.

**Syntax:**

List l1=new List(4);

l1.setBounds(100,100, 75,75);

**6. Text Field**

A text field is used for the editing of a particular line of text which can be used within the programming concept.

**Syntax:**

TextField na=new TextField(20);

**7. Text Area**

A text area is used for the editing of multiple lines of text. The only difference between the Text field and Text area is that Text Field is used for editing a single line of text within the user interface while a Text Area is used for editing multiple lines of text.

**Syntax:**

TextArea area=new TextArea("Welcome to the universe");

area.setBounds(10,30, 300,300);

**8. Choice**

A choice, as the name implies, shows the various options and the choice that is selected is shown in the top menu bar of the screen.

**Syntax:**

Choice c=new Choice();

c.setBounds(100,100, 75,75);

c.add("Subject 1");

c.add("Subject 2");

c.add("Subject 3");

c.add("Subject 4");

c.add("Subject 5");

**9. Canvas**

In the canvas space, there can be an input being given by the user or the user can draw something on the Canvas space being given.

**Syntax:**

f.add(new MyCanvas());

f.setLayout(null);

f.setSize(400, 400);

f.setVisible(true);

**10. Image**

There can be a single image or multiple images within a UI. There can be a button being associated with an image and when it is clicked it can produce some functionality.

**Syntax:**

Image i=t.getImage("pic2.gif");

**11. Scroll Bar**

The scroll bar like a normal one is used to scroll or move from a varied range of values. The user selects one value from those range of values.

**Syntax:**

Scrollbar s=new Scrollbar();

s.setBounds(100,100, 50,100);

**12. Dialog**

The dialog is used to take some form of input from the user and produce it in a sequential manner.

**Syntax:**

d = new Dialog(f , "Hello World", true);

**13. File Dialog**

From a file dialog, a user can select a file which he/she wishes to use.

**Syntax:**

FileDialog(Dialog parent)

**Example:**

import java.awt.\*;

**public** **class** AwtApp **extends** Frame

{

AwtApp()

{

Label firstName = **new** Label("First Name");

firstName.setBounds(20, 50, 80, 20);

Label lastName = **new** Label("Last Name");

lastName.setBounds(20, 80, 80, 20);

Label dob = **new** Label("Date of Birth");

dob.setBounds(20, 110, 80, 20);

TextField firstNameTF = **new** TextField();

firstNameTF.setBounds(120, 50, 100, 20);

TextField lastNameTF = **new** TextField();

lastNameTF.setBounds(120, 80, 100, 20);

TextField dobTF = **new** TextField();

dobTF.setBounds(120, 110, 100, 20);

Button sbmt = **new** Button("Submit");

sbmt.setBounds(20, 160, 100, 30);

Button reset = **new** Button("Reset");

reset.setBounds(120,160,100,30);

add(firstName);

add(lastName);

add(dob);

add(firstNameTF);

add(lastNameTF);

add(dobTF);

add(sbmt);

add(reset);

setSize(300,300);

setLayout(**null**);

setVisible(**true**);

}

**public** **static** **void** main(String[] args)

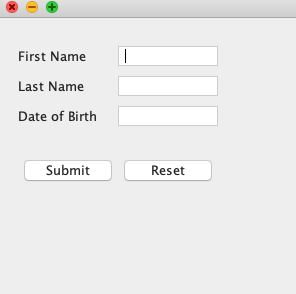
{

AwtApp awt = **new** AwtApp();

}

}

**Output:**



**14. Menu Bars and Menus**

A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu. This concept is implemented in the AWT by the following classes: **MenuBar**, **Menu**, and **MenuItem**. In general, a menu bar contains one or more **Menu** objects. Each **Menu** object contains a list of **MenuItem** objects. Each **MenuItem** object represents something that can be selected by the user. Since **Menu** is a subclass of **MenuItem**, a hierarchy of nested submenus can be created.

import java.awt.\*;

class MenuExample

{

     MenuExample(){

         Frame f= new Frame("Menu and MenuItem Example");

         MenuBar mb=new MenuBar();

         Menu menu=new Menu("Menu");

         Menu submenu=new Menu("Sub Menu");

         MenuItem i1=new MenuItem("Item 1");

         MenuItem i2=new MenuItem("Item 2");

         MenuItem i3=new MenuItem("Item 3");

         MenuItem i4=new MenuItem("Item 4");

         MenuItem i5=new MenuItem("Item 5");

         menu.add(i1);

         menu.add(i2);

         menu.add(i3);

         submenu.add(i4);

         submenu.add(i5);

         menu.add(submenu);

         mb.add(menu);

         f.setMenuBar(mb);

         f.setSize(400,400);

         f.setLayout(null);

         f.setVisible(true);

}

public static void main(String args[])

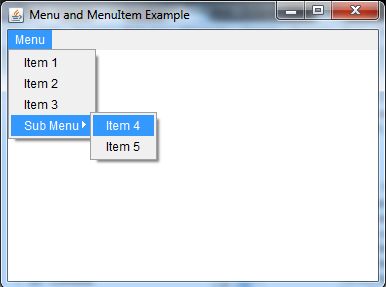
{

new MenuExample();

}

}

Output:



**Types of Layout Manager in Java**

In Java, graphical user interfaces (GUIs) play a vital role in creating interactive applications. To design a visually appealing and organized interface, the choice of layout manager becomes crucial. **Layout managers define how components are arranged within a container, such as a Frame or Panel.** Java provides several layout managers to suit various design needs. In this section, we will delve into the details of the different types of layout managers available in Java, along with code examples and explanations.

>>The **pack()**method is defined in **Window**class in Java and it sizes the frame so that all its contents are at or above their preferred sizes. An alternative to the **pack()**method is to establish a frame size explicitly by calling the **setSize()** or **setBounds()** methods. In general, using the **pack()** method is preferable to call than **setSize()**method, since pack leaves the frame layout manager in charge of the frame size and layout managers are good at adjusting to platform dependencies and other factors that affect the component size.

**Syntax**

public void pack()

**1. FlowLayout**

FlowLayout is a simple layout manager that arranges components in a row, left to right, wrapping to the next line as needed. It is ideal for scenarios where components need to maintain their natural sizes and maintain a flow-like structure.

FlowLayoutExample.java

import javax.swing.\*;

import java.awt.\*;

public class FlowLayoutExample {

    public static void main(String[] args) {

        JFrame frame = new JFrame("FlowLayout Example");

        frame.setLayout(new FlowLayout());

        frame.add(new JButton("Button 1"));

        frame.add(new JButton("Button 2"));

        frame.add(new JButton("Button 3"));

        frame.pack();

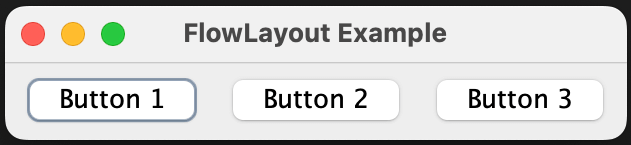
        frame.setVisible(true);

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    }

}

Output:



**2. BorderLayout**

BorderLayout divides the container into five regions: NORTH, SOUTH, EAST, WEST, and CENTER. Components can be added to these regions, and they will occupy the available space accordingly. This layout manager is suitable for creating interfaces with distinct sections, such as a title bar, content area, and status bar.

BorderLayoutExample.java

import javax.swing.\*;

import java.awt.\*;

public class BorderLayoutExample {

    public static void main(String[] args) {

        JFrame frame = new JFrame("BorderLayout Example");

        frame.setLayout(new BorderLayout());

        frame.add(new JButton("North"), BorderLayout.NORTH);

        frame.add(new JButton("South"), BorderLayout.SOUTH);

        frame.add(new JButton("East"), BorderLayout.EAST);

        frame.add(new JButton("West"), BorderLayout.WEST);

        frame.add(new JButton("Center"), BorderLayout.CENTER);

        frame.pack();

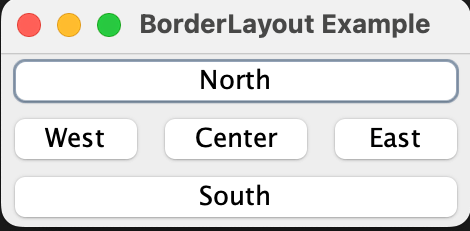
        frame.setVisible(true);

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    }

}

Output:



**3. GridLayout**

GridLayout arranges components in a grid with a specified number of rows and columns. Each cell in the grid can hold a component. This layout manager is ideal for creating a uniform grid of components, such as a calculator or a game board.

GridLayoutExample.java

import javax.swing.\*;

import java.awt.\*;

public class GridLayoutExample {

    public static void main(String[] args) {

        JFrame frame = new JFrame("GridLayout Example");

        frame.setLayout(new GridLayout(3, 3));

        for (int i = 1; i <= 9; i++) {

            frame.add(new JButton("Button " + i));

        }

        frame.pack();

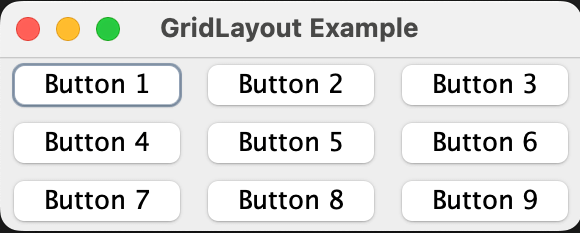
        frame.setVisible(true);

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    }

}

Output:



**4. CardLayout**

CardLayout allows components to be stacked on top of each other, like a deck of cards. Only one component is visible at a time, and you can switch between components using methods like next() and previous(). This layout is useful for creating wizards or multi-step processes.

CardLayoutExample.java

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class CardLayoutExample {

    public static void main(String[] args) {

        JFrame frame = new JFrame("CardLayout Example");

        CardLayout cardLayout = new CardLayout();

        JPanel cardPanel = new JPanel(cardLayout);

        JButton button1 = new JButton("Card 1");

        JButton button2 = new JButton("Card 2");

        JButton button3 = new JButton("Card 3");

        cardPanel.add(button1, "Card 1");

        cardPanel.add(button2, "Card 2");

        cardPanel.add(button3, "Card 3");

        frame.add(cardPanel);

        frame.pack();

        frame.setVisible(true);

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        button1.addActionListener(e -> cardLayout.show(cardPanel, "Card 2"));

        button2.addActionListener(e -> cardLayout.show(cardPanel, "Card 3"));

        button3.addActionListener(e -> cardLayout.show(cardPanel, "Card 1"));

    }

}

Output:

