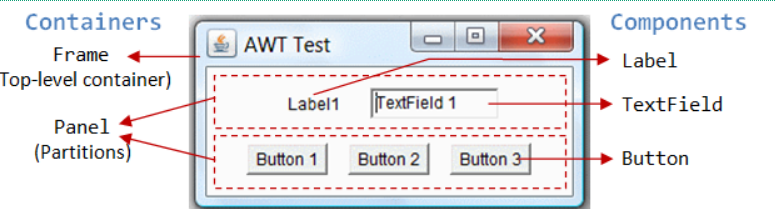
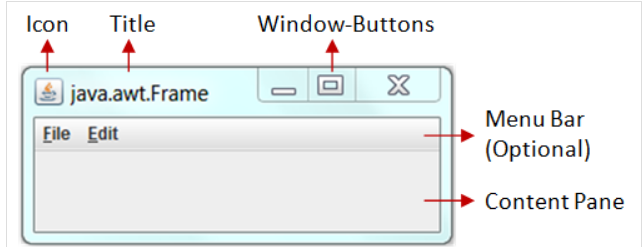
## Add GUI plug in Eclipse –link:: https://www.youtube.com/watch?v=oeswfZz4IW0

## What is GUI in Java?

**GUI (Graphical User Interface) in Java** is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.





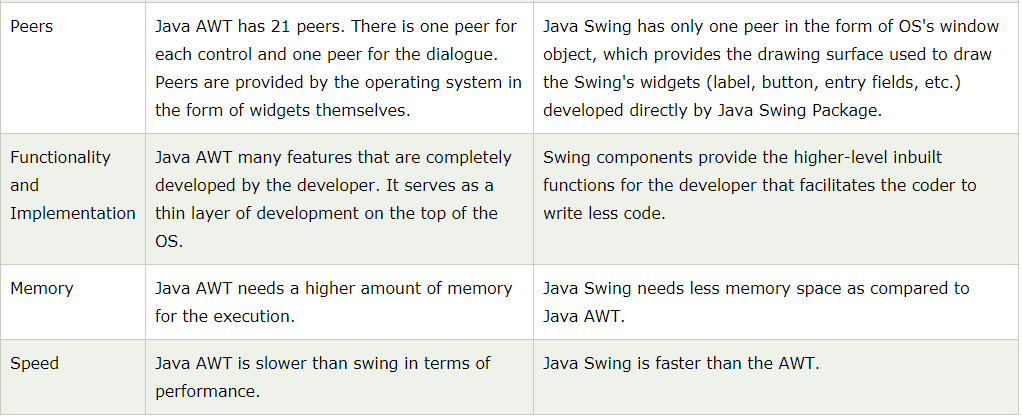
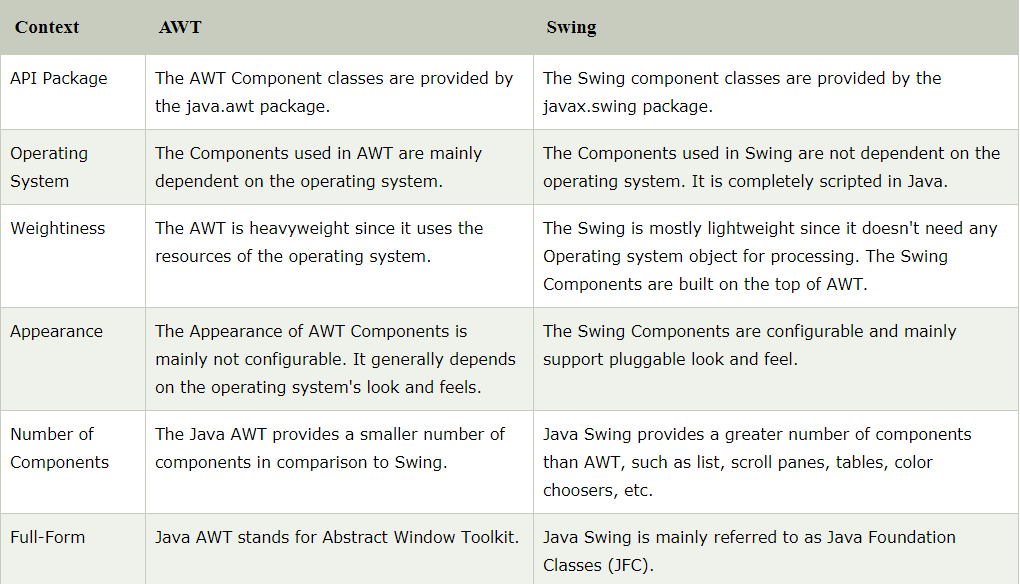
AWT and Swing in Java

AWT and Swing are used to develop window-based applications in Java.

Awt is an abstract window toolkit that provides various component classes like Label, Button, TextField, etc., to show window components on the screen. All these classes are part of the Java.awt package.

Swing is the part of JFC (Java Foundation Classes) built on the top of AWT and written entirely in [Java](https://www.javatpoint.com/java-tutorial). The javax.swing API provides all the component classes like JButton, JTextField, JCheckbox, JMenu, etc.

The components of Swing are platform-independent, i.e., swing doesn't depend on the operating system to show the components. Also, the Swing's components are lightweight. The main differences between [AWT](https://www.javatpoint.com/java-awt) and [Swing](https://www.javatpoint.com/java-swing) are given in the following table.



**Java AWT**

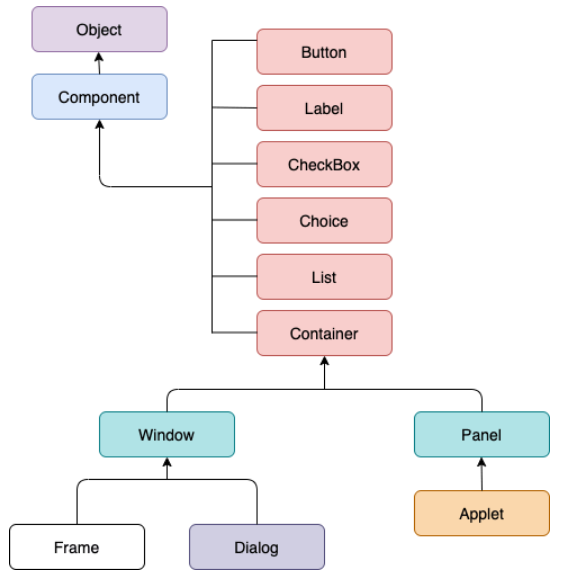
**Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

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](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.

### **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](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.

### **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.

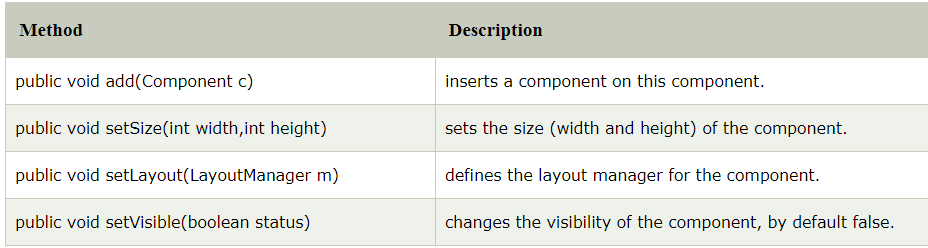
### **Panel**

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.

### **Useful Methods of Component class**



### **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**

* 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

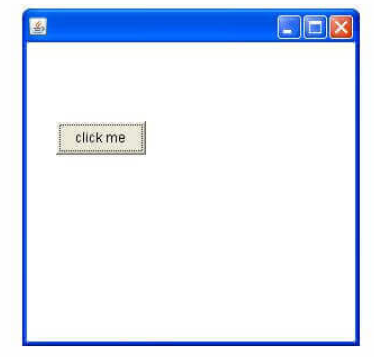
}

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

First f=**new** First();

}}

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**

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**);

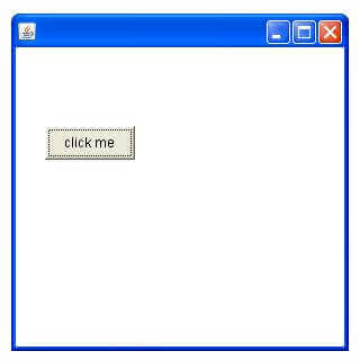
f.setVisible(**true**);

}

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

First2 f=**new** First2();

}}



# Java AWT Button

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

## **AWT Button Class declaration**

**public** **class** Button **extends** Component **implements** Accessible

## **Java AWT Button Example**

**import** java.awt.\*;

**public** **class** ButtonExample {

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

    Frame f=**new** Frame("Button Example");

    Button b=**new** Button("Click Here");

    b.setBounds(50,100,80,30);

    f.add(b);

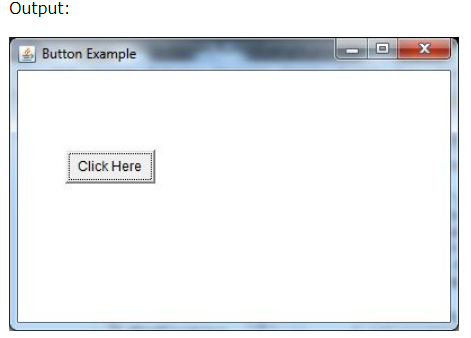
    f.setSize(400,400);

    f.setLayout(**null**);

    f.setVisible(**true**);

}

}



# Java AWT Label

The [object](https://www.javatpoint.com/object-and-class-in-java) of Label 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.

## **AWT Label Class Declaration**

**public** **class** Label **extends** Component **implements** Accessible

**Java Label Example**

**import** java.awt.\*;

**class** LabelExample{

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

    Frame f= **new** Frame("Label Example");

    Label l1,l2;

    l1=**new** Label("First Label.");

    l1.setBounds(50,100, 100,30);

    l2=**new** Label("Second Label.");

    l2.setBounds(50,150, 100,30);

    f.add(l1); f.add(l2);

    f.setSize(400,400);

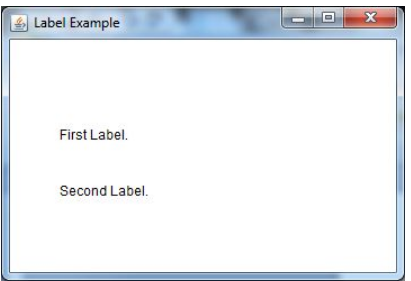
    f.setLayout(**null**);

    f.setVisible(**true**);

}

}

Output:



# Java AWT TextField

The [object](https://www.javatpoint.com/object-and-class-in-java) of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

## **AWT TextField Class Declaration**

**public** **class** TextField **extends** TextComponent

## **Java AWT TextField Example**

**import** java.awt.\*;

**class** TextFieldExample{

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

    Frame f= **new** Frame("TextField Example");

    TextField t1,t2;

    t1=**new** TextField("Welcome to VIT.");

    t1.setBounds(50,100, 200,30);

    t2=**new** TextField("AWT Tutorial");

    t2.setBounds(50,150, 200,30);

    f.add(t1); f.add(t2);

    f.setSize(400,400);

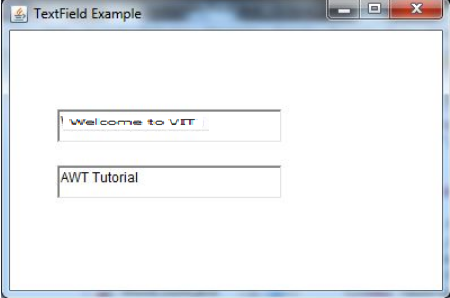
    f.setLayout(**null**);

    f.setVisible(**true**);

}

}

Ouput:



# Java AWT TextArea

The [object](https://www.javatpoint.com/object-and-class-in-java) of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

## **AWT TextArea Class Declaration**

**public** **class** TextArea **extends** TextComponent

## **Java AWT TextArea Example**

**import** java.awt.\*;

**public** **class** TextAreaExample

{

     TextAreaExample(){

        Frame f= **new** Frame();

            TextArea area=**new** TextArea("Welcome to VIT ");

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

        f.add(area);

        f.setSize(400,400);

        f.setLayout(**null**);

        f.setVisible(**true**);

     }

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

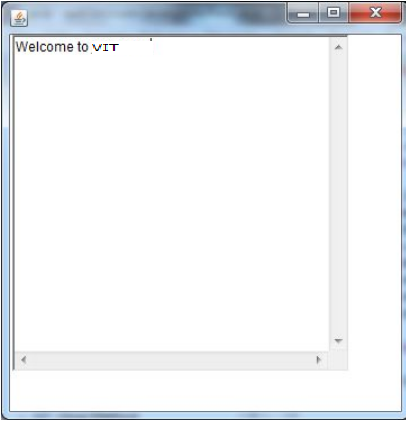
{

**new** TextAreaExample();

}

}

Output:



# Java AWT Choice

The object of Choice class is used to show [popup menu](https://www.javatpoint.com/java-awt-popupmenu) of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

## **AWT Choice Class Declaration**

**public** **class** Choice **extends** Component **implements** ItemSelectable, Accessible

## **Java AWT Choice Example**

**import** java.awt.\*;

**public** **class** ChoiceExample

{

        ChoiceExample(){

        Frame f= **new** Frame();

        Choice c=**new** Choice();

        c.setBounds(100,100, 75,75);

        c.add("Item 1");

        c.add("Item 2");

        c.add("Item 3");

        c.add("Item 4");

        c.add("Item 5");

        f.add(c);

        f.setSize(400,400);

        f.setLayout(**null**);

        f.setVisible(**true**);

     }

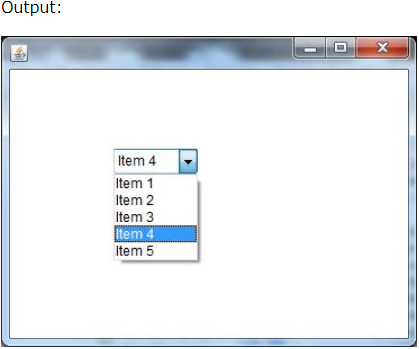
**public** **static** **void** main(String args[])

{

**new** ChoiceExample();

}

}



# Java AWT List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

## **AWT List class Declaration**

**public** **class** List **extends** Component **implements** ItemSelectable, Accessible

## **Java AWT List Example**

**import** java.awt.\*;

**public** **class** ListExample

{

     ListExample(){

        Frame f= **new** Frame();

        List l1=**new** List(5);

        l1.setBounds(100,100, 75,75);

        l1.add("Item 1");

        l1.add("Item 2");

        l1.add("Item 3");

        l1.add("Item 4");

        l1.add("Item 5");

        f.add(l1);

        f.setSize(400,400);

        f.setLayout(**null**);

        f.setVisible(**true**);

     }

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

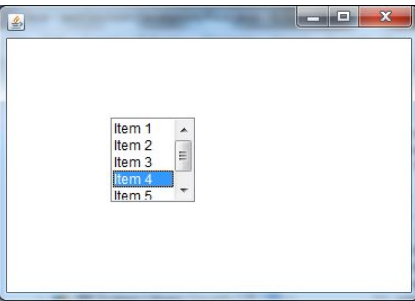
{

**new** ListExample();

}

}

Output:



# Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

## **AWT MenuItem class declaration**

**public** **class** MenuItem **extends** MenuComponent **implements** Accessible

## **AWT Menu class declaration**

**public** **class** Menu **extends** MenuItem **implements** MenuContainer, Accessible

## **Java AWT MenuItem and Menu Example**

**import** java.awt.\*;

**class** MenuExample

{

     MenuExample(){

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

         MenuBar mb=**new** MenuBar();

         Menu menu=**new** Menu("Menu");

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

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

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

         menu.add(i1);

         menu.add(i2);

         menu.add(i3);

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

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

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

         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[])

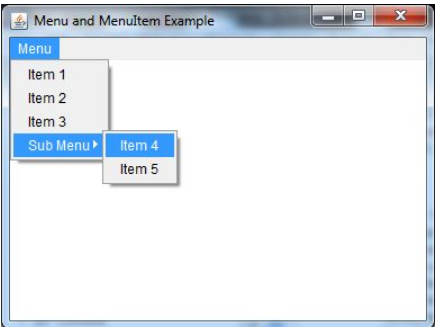
{

**new** MenuExample();

}

}

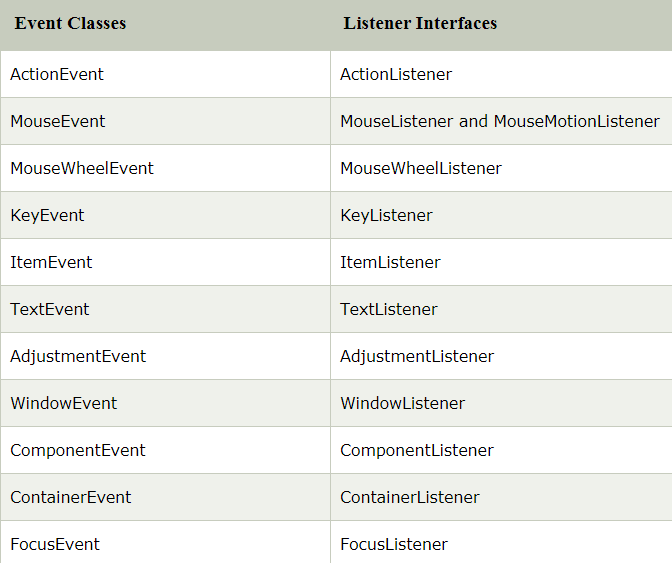
Output:



# 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**



## **Steps to perform Event Handling**

Following steps are required to perform event handling:

1. Register the component with the Listener

## **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**

// The ActionListener interface is found in java.awt.event [package](https://www.javatpoint.com/package). It has only one method: actionPerformed().

**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");

}

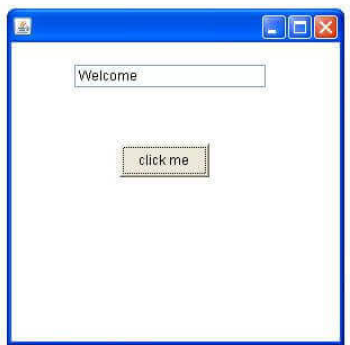
**public** **static** **void** main(String args[]){

**new** AEvent();

}

}

**public void setBounds(int xaxis, int yaxis, int width, int height);** have been used in the above example that sets the position of the component it may be button, textfield etc.



### **2) 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();

}

}

**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");

}

}

### **3) 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()

{

tf.setText("hello");

}

});

add(b);add(tf);

setSize(300,300);

setLayout(**null**);

setVisible(**true**);

}

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

**new** AEvent3();

}

}

## **Java AWT Button Example with ActionListener**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** ButtonExample {

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

    Frame f=**new** Frame("Button Example");

**final** TextField tf=**new** TextField();

    tf.setBounds(50,50, 150,20);

    Button b=**new** Button("Click Here");

    b.setBounds(50,100,60,30);

    b.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e){

            tf.setText("Welcome to VIT");

        }

    });

    f.add(b);f.add(tf);

    f.setSize(400,400);

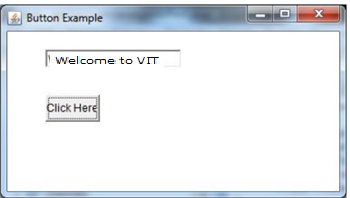
    f.setLayout(**null**);

    f.setVisible(**true**);

}

}

Output:



## **Java AWT Label Example with ActionListener**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** LabelExample **extends** Frame **implements** ActionListener{

    TextField tf; Label l; Button b;

    LabelExample(){

        tf=**new** TextField();

        tf.setBounds(50,50, 150,20);

        l=**new** Label();

        l.setBounds(50,100, 250,20);

        b=**new** Button("Find IP");

        b.setBounds(50,150,60,30);

        b.addActionListener(**this**);

        add(b);add(tf);add(l);

        setSize(400,400);

        setLayout(**null**);

        setVisible(**true**);

    }

**public** **void** actionPerformed(ActionEvent e) {

**try**{

        String host=tf.getText();

        String ip=java.net.InetAddress.getByName(host).getHostAddress();

        l.setText("IP of "+host+" is: "+ip);

        }**catch**(Exception ex){System.out.println(ex);}

    }

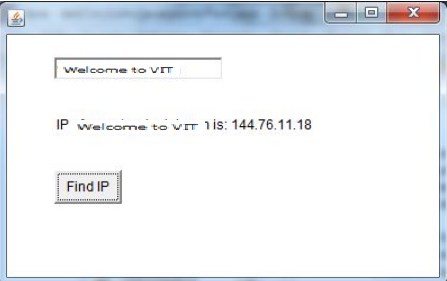
**public** **static** **void** main(String[] args) {

**new** LabelExample();

    }

}

Output:



## **Java AWT TextField Example with ActionListener**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** TextFieldExample **extends** Frame **implements** ActionListener{

    TextField tf1,tf2,tf3;

    Button b1,b2;

    TextFieldExample(){

        tf1=**new** TextField();

        tf1.setBounds(50,50,150,20);

        tf2=**new** TextField();

        tf2.setBounds(50,100,150,20);

        tf3=**new** TextField();

        tf3.setBounds(50,150,150,20);

        tf3.setEditable(**false**);

        b1=**new** Button("+");

        b1.setBounds(50,200,50,50);

        b2=**new** Button("-");

        b2.setBounds(120,200,50,50);

        b1.addActionListener(**this**);

        b2.addActionListener(**this**);

        add(tf1);add(tf2);add(tf3);add(b1);add(b2);

        setSize(300,300);

        setLayout(**null**);

        setVisible(**true**);

    }

**public** **void** actionPerformed(ActionEvent e) {

        String s1=tf1.getText();

        String s2=tf2.getText();

**int** a=Integer.parseInt(s1);

**int** b=Integer.parseInt(s2);

**int** c=0;

**if**(e.getSource()==b1){

            c=a+b;

        }**else** **if**(e.getSource()==b2){

            c=a-b;

        }

        String result=String.valueOf(c);

        tf3.setText(result);

    }

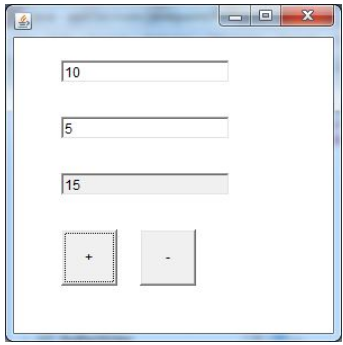
**public** **static** **void** main(String[] args) {

**new** TextFieldExample();

}

}

Output:



## **Java AWT TextArea Example with ActionListener**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** TextAreaExample **extends** Frame **implements** ActionListener{

Label l1,l2;

TextArea area;

Button b;

TextAreaExample(){

    l1=**new** Label();

    l1.setBounds(50,50,100,30);

    l2=**new** Label();

    l2.setBounds(160,50,100,30);

    area=**new** TextArea();

    area.setBounds(20,100,300,300);

    b=**new** Button("Count Words");

    b.setBounds(100,400,100,30);

    b.addActionListener(**this**);

    add(l1);add(l2);add(area);add(b);

    setSize(400,450);

    setLayout(**null**);

    setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e){

    String text=area.getText();

    String words[]=text.split("\\s");

    l1.setText("Words: "+words.length);

    l2.setText("Characters: "+text.length());

}

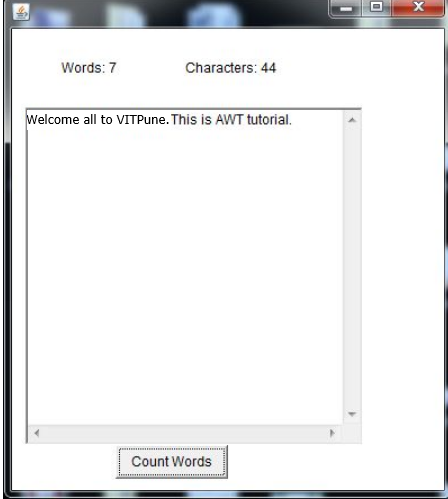
**public** **static** **void** main(String[] args) {

**new** TextAreaExample();

}

}

Output:



## **Java AWT Choice Example with ActionListener**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** ChoiceExample

{

        ChoiceExample(){

        Frame f= **new** Frame();

**final** Label label = **new** Label();

        label.setAlignment(Label.CENTER);

        label.setSize(400,100);

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

        b.setBounds(200,100,50,20);

**final** Choice c=**new** Choice();

        c.setBounds(100,100, 75,75);

        c.add("C");

        c.add("C++");

        c.add("Java");

        c.add("PHP");

        c.add("Android");

        f.add(c);f.add(label); f.add(b);

        f.setSize(400,400);

        f.setLayout(**null**);

        f.setVisible(**true**);

        b.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

         String data = "Programming language Selected: "+ c.getItem(c.getSelectedIndex());

         label.setText(data);

        }

        });

        }

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

{

**new** ChoiceExample();

}

}

Output:



## **Java AWT List Example with ActionListener**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** ListExample

{

     ListExample(){

        Frame f= **new** Frame();

**final** Label label = **new** Label();

        label.setAlignment(Label.CENTER);

        label.setSize(500,100);

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

        b.setBounds(200,150,80,30);

**final** List l1=**new** List(4, **false**);

        l1.setBounds(100,100, 70,70);

        l1.add("C");

        l1.add("C++");

        l1.add("Java");

        l1.add("PHP");

**final** List l2=**new** List(4, **true**);

        l2.setBounds(100,200, 70,70);

        l2.add("Turbo C++");

        l2.add("Spring");

        l2.add("Hibernate");

        l2.add("CodeIgniter");

        f.add(l1); f.add(l2); f.add(label); f.add(b);

        f.setSize(450,450);

        f.setLayout(**null**);

        f.setVisible(**true**);

        b.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

          String data = "Programming language Selected: "+l1.getItem(l1.getSelectedIndex());

          data += ", Framework Selected:";

**for**(String frame:l2.getSelectedItems()){

                   data += frame + " ";

          }

          label.setText(data);

          }

         });

}

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

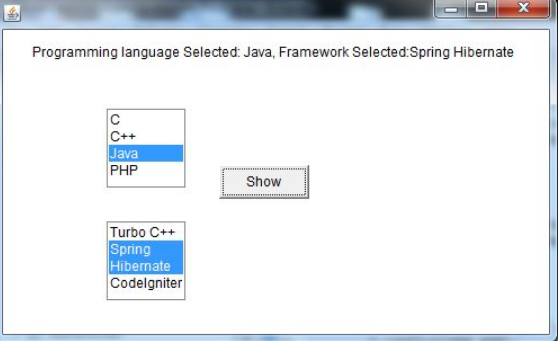
{

**new** ListExample();

}

}

Output:



# Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

## **Methods of MouseListener interface**

The signature of 5 methods found in MouseListener interface are given below:

1. **public** **abstract** **void** mouseClicked(MouseEvent e);
2. **public** **abstract** **void** mouseEntered(MouseEvent e);
3. **public** **abstract** **void** mouseExited(MouseEvent e);
4. **public** **abstract** **void** mousePressed(MouseEvent e);
5. **public** **abstract** **void** mouseReleased(MouseEvent e);

## **Java MouseListener Example**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** MouseListenerExample **extends** Frame **implements** MouseListener{

    Label l;

    MouseListenerExample(){

        addMouseListener(**this**);

        l=**new** Label();

        l.setBounds(20,50,100,20);

        add(l);

        setSize(300,300);

        setLayout(**null**);

        setVisible(**true**);

    }

**public** **void** mouseClicked(MouseEvent e) {

        l.setText("Mouse Clicked");

    }

**public** **void** mouseEntered(MouseEvent e) {

        l.setText("Mouse Entered");

    }

**public** **void** mouseExited(MouseEvent e) {

        l.setText("Mouse Exited");

    }

**public** **void** mousePressed(MouseEvent e) {

        l.setText("Mouse Pressed");

    }

**public** **void** mouseReleased(MouseEvent e) {

        l.setText("Mouse Released");

    }

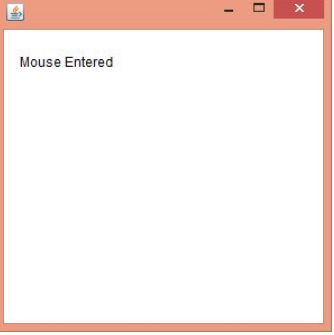
**public** **static** **void** main(String[] args) {

**new** MouseListenerExample();

}

}

Output:



## **Java MouseListener Example 2**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** MouseListenerExample2 **extends** Frame **implements** MouseListener{

    MouseListenerExample2(){

        addMouseListener(**this**);

        setSize(300,300);

        setLayout(**null**);

        setVisible(**true**);

    }

**public** **void** mouseClicked(MouseEvent e) {

        Graphics g=getGraphics();

        g.setColor(Color.BLUE);

        g.fillOval(e.getX(),e.getY(),30,30);

    }

**public** **void** mouseEntered(MouseEvent e) {}

**public** **void** mouseExited(MouseEvent e) {}

**public** **void** mousePressed(MouseEvent e) {}

**public** **void** mouseReleased(MouseEvent e) {}

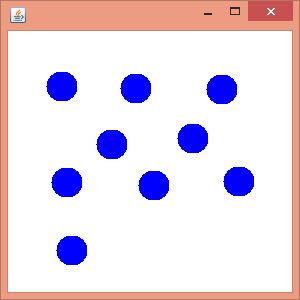
**public** **static** **void** main(String[] args) {

**new** MouseListenerExample2();

}

}

Output:



# Java MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

## **Methods of MouseMotionListener interface**

The signature of 2 methods found in MouseMotionListener interface are given below:

1. **public** **abstract** **void** mouseDragged(MouseEvent e);
2. **public** **abstract** **void** mouseMoved(MouseEvent e);

## **Java MouseMotionListener Example**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** MouseMotionListenerExample **extends** Frame **implements** MouseMotionListener{

    MouseMotionListenerExample(){

        addMouseMotionListener(**this**);

        setSize(300,300);

        setLayout(**null**);

        setVisible(**true**);

    }

**public** **void** mouseDragged(MouseEvent e) {

    Graphics g=getGraphics();

    g.setColor(Color.BLUE);

    g.fillOval(e.getX(),e.getY(),20,20);

}

**public** **void** mouseMoved(MouseEvent e) {}

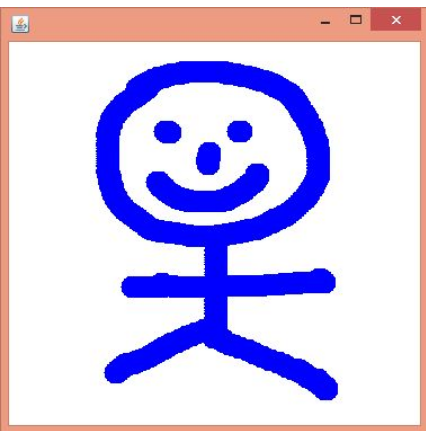
**public** **static** **void** main(String[] args) {

**new** MouseMotionListenerExample();

}

}

Output:



# Java KeyListener Interface

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.

## **Methods of KeyListener interface**

The signature of 3 methods found in KeyListener interface are given below:

1. **public** **abstract** **void** keyPressed(KeyEvent e);
2. **public** **abstract** **void** keyReleased(KeyEvent e);
3. **public** **abstract** **void** keyTyped(KeyEvent e);

## **Java KeyListener Example**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** KeyListenerExample **extends** Frame **implements** KeyListener{

    Label l;

    TextArea area;

    KeyListenerExample(){

        l=**new** Label();

        l.setBounds(20,50,100,20);

        area=**new** TextArea();

        area.setBounds(20,80,300, 300);

        area.addKeyListener(**this**);

        add(l);add(area);

        setSize(400,400);

        setLayout(**null**);

        setVisible(**true**);

    }

**public** **void** keyPressed(KeyEvent e) {

        l.setText("Key Pressed");

    }

**public** **void** keyReleased(KeyEvent e) {

        l.setText("Key Released");

    }

**public** **void** keyTyped(KeyEvent e) {

        l.setText("Key Typed");

    }

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

**new** KeyListenerExample();

    }

}

Output:



## **Java KeyListener Example 2: Count Words & Characters**

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** KeyListenerExample **extends** Frame **implements** KeyListener{

    Label l;

    TextArea area;

    KeyListenerExample(){

        l=**new** Label();

        l.setBounds(20,50,200,20);

        area=**new** TextArea();

        area.setBounds(20,80,300, 300);

        area.addKeyListener(**this**);

        add(l);add(area);

        setSize(400,400);

        setLayout(**null**);

        setVisible(**true**);

    }

**public** **void** keyPressed(KeyEvent e) {}

**public** **void** keyReleased(KeyEvent e) {

        String text=area.getText();

        String words[]=text.split("\\s");

        l.setText("Words: "+words.length+" Characters:"+text.length());

    }

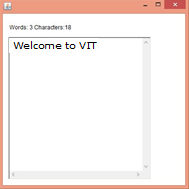
**public** **void** keyTyped(KeyEvent e) {}

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

**new** KeyListenerExample();

    }  }

Output:



# 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.

# Java Swing Tutorial

**Java Swing tutorial** 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**

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** |

### **What is JFC**

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

### **Hierarchy of Java Swing classes**

The hierarchy of java swing API is given below.

# hierarchy of javax swing

### **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

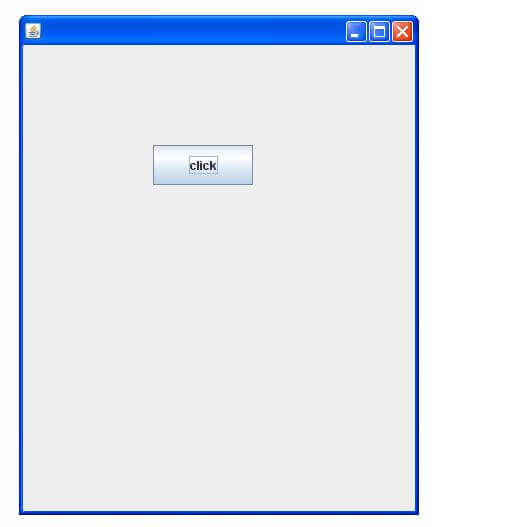
f.setSize(400,500);//400 width and 500 height

f.setLayout(**null**);//using no layout managers

f.setVisible(**true**);//making the frame visible

}

}



### **Example of Swing by Association inside constructor**

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

*File: Simple.java*

**import** javax.swing.\*;

**public** **class** Simple {

JFrame f;

Simple(){

f=**new** JFrame();//creating instance of JFrame

JButton b=**new** JButton("click");//creating instance of JButton

b.setBounds(130,100,100, 40);

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

}

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

**new** Simple();

}

}

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

### **Simple example of Swing by inheritance**

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

*File: Simple2.java*

**import** javax.swing.\*;

**public** **class** Simple2 **extends** JFrame{//inheriting JFrame

JFrame f;

Simple2(){

JButton b=**new** JButton("click");//create button

b.setBounds(130,100,100, 40);

add(b);//adding button on frame

setSize(400,500);

setLayout(**null**);

setVisible(**true**);

}

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

**new** Simple2();

}}

*What we will learn in Swing*

* JButton class
* JRadioButton class
* JTextArea class
* JComboBox class
* JTable class
* JColorChooser class
* JProgressBar class
* JSlider class
* Digital Watch
* Graphics in swing
* Displaying image
* Edit menu code for Notepad
* OpenDialog Box
* Notepad
* Puzzle Game
* Pic Puzzle Game
* Tic Tac Toe Game
* BorderLayout
* GridLayout
* FlowLayout
* CardLayout

# Java 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. It inherits AbstractButton class.

## **JButton class declaration**

Let's see the declaration for javax.swing.JButton class.

**public** **class** JButton **extends** AbstractButton **implements** Accessible

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JButton() | It creates a button with no text and icon. |
| JButton(String s) | It creates a button with the specified text. |
| JButton(Icon i) | It creates a button with the specified icon object. |

### **Commonly used Methods of AbstractButton class:**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void setText(String s) | It is used to set specified text on button |
| String getText() | It is used to return the text of the button. |
| void setEnabled(boolean b) | It is used to enable or disable the button. |
| void setIcon(Icon b) | It is used to set the specified Icon on the button. |
| Icon getIcon() | It is used to get the Icon of the button. |
| void setMnemonic(int a) | It is used to set the mnemonic on the button. |
| void addActionListener(ActionListener a) | It is used to add the [action listener](https://www.javatpoint.com/java-actionlistener) to this object. |

## **Java JButton Example**

**import** javax.swing.\*;

**public** **class** ButtonExample {

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

    JFrame f=**new** JFrame("Button Example");

    JButton b=**new** JButton("Click Here");

    b.setBounds(50,100,95,30);

    f.add(b);

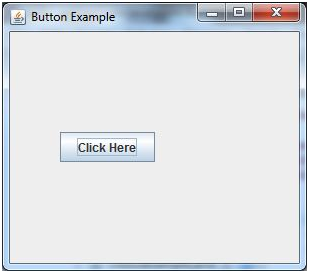
    f.setSize(400,400);

    f.setLayout(**null**);

    f.setVisible(**true**);

}

}  Output:



## **Java JButton Example with ActionListener**

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** ButtonExample {

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

    JFrame f=**new** JFrame("Button Example");

**final** JTextField tf=**new** JTextField();

    tf.setBounds(50,50, 150,20);

    JButton b=**new** JButton("Click Here");

    b.setBounds(50,100,95,30);

    b.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e){

            tf.setText("Welcome to Javatpoint.");

        }

    });

    f.add(b);f.add(tf);

    f.setSize(400,400);

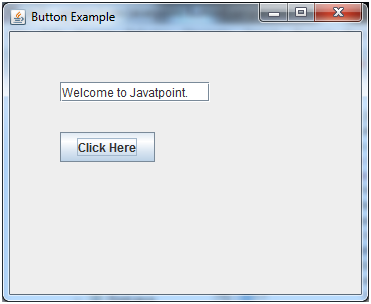
    f.setLayout(**null**);

    f.setVisible(**true**);

}

}

Output:



## **Example of displaying image on the button:**

**import** javax.swing.\*;

**public** **class** ButtonExample{

ButtonExample(){

JFrame f=**new** JFrame("Button Example");

JButton b=**new** JButton(**new** ImageIcon("D:\\icon.png"));

b.setBounds(100,100,100, 40);

f.add(b);

f.setSize(300,400);

f.setLayout(**null**);

f.setVisible(**true**);

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

    }

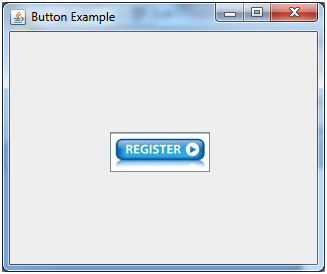
**public** **static** **void** main(String[] args) {

**new** ButtonExample();

}

}

Output:



# Java 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. It inherits JComponent class.

## **JLabel class declaration**

Let's see the declaration for javax.swing.JLabel class.

1. **public** **class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JLabel() | Creates a JLabel instance with no image and with an empty string for the title. |
| JLabel(String s) | Creates a JLabel instance with the specified text. |
| JLabel(Icon i) | Creates a JLabel instance with the specified image. |
| JLabel(String s, Icon i, int horizontalAlignment) | Creates a JLabel instance with the specified text, image, and horizontal alignment. |

### **Commonly used Methods:**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| String getText() | it returns the text string that a label displays. |
| void setText(String text) | It defines the single line of text this component will display. |
| void setHorizontalAlignment(int alignment) | It sets the alignment of the label's contents along the X axis. |
| Icon getIcon() | It returns the graphic image that the label displays. |
| int getHorizontalAlignment() | It returns the alignment of the label's contents along the X axis. |

## **Java JLabel Example**

**import** javax.swing.\*;

**class** LabelExample

{

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

    {

    JFrame f= **new** JFrame("Label Example");

    JLabel l1,l2;

    l1=**new** JLabel("First Label.");

    l1.setBounds(50,50, 100,30);

    l2=**new** JLabel("Second Label.");

    l2.setBounds(50,100, 100,30);

    f.add(l1); f.add(l2);

    f.setSize(300,300);

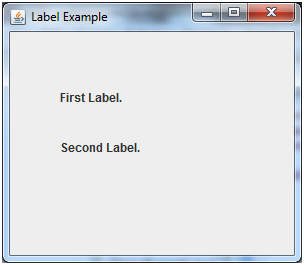
    f.setLayout(**null**);

    f.setVisible(**true**);

    }

    }

Output:



## **Java JLabel Example with ActionListener**

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

**public** **class** LabelExample **extends** Frame **implements** ActionListener{

    JTextField tf; JLabel l; JButton b;

    LabelExample(){

        tf=**new** JTextField();

        tf.setBounds(50,50, 150,20);

        l=**new** JLabel();

        l.setBounds(50,100, 250,20);

        b=**new** JButton("Find IP");

        b.setBounds(50,150,95,30);

        b.addActionListener(**this**);

        add(b);add(tf);add(l);

        setSize(400,400);

        setLayout(**null**);

        setVisible(**true**);

    }

**public** **void** actionPerformed(ActionEvent e) {

**try**{

        String host=tf.getText();

        String ip=java.net.InetAddress.getByName(host).getHostAddress();

        l.setText("IP of "+host+" is: "+ip);

        }**catch**(Exception ex){System.out.println(ex);}

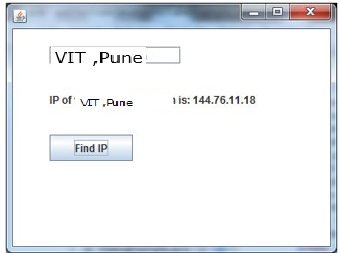
    }

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

**new** LabelExample();

    } }

Output:



# Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

## **JTextField class declaration**

Let's see the declaration for javax.swing.JTextField class.

**public** **class** JTextField **extends** JTextComponent **implements** SwingConstants

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JTextField() | Creates a new TextField |
| JTextField(String text) | Creates a new TextField initialized with the specified text. |
| JTextField(String text, int columns) | Creates a new TextField initialized with the specified text and columns. |
| JTextField(int columns) | Creates a new empty TextField with the specified number of columns. |

### **Commonly used Methods:**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void addActionListener(ActionListener l) | It is used to add the specified action listener to receive action events from this textfield. |
| Action getAction() | It returns the currently set Action for this ActionEvent source, or null if no Action is set. |
| void setFont(Font f) | It is used to set the current font. |
| void removeActionListener(ActionListener l) | It is used to remove the specified action listener so that it no longer receives action events from this textfield. |

## **Java JTextField Example**

**import** javax.swing.\*;

**class** TextFieldExample

{

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

    {

    JFrame f= **new** JFrame("TextField Example");

    JTextField t1,t2;

    t1=**new** JTextField("Welcome to VIT,Pune.");

    t1.setBounds(50,100, 200,30);

    t2=**new** JTextField("AWT Tutorial");

    t2.setBounds(50,150, 200,30);

    f.add(t1); f.add(t2);

    f.setSize(400,400);

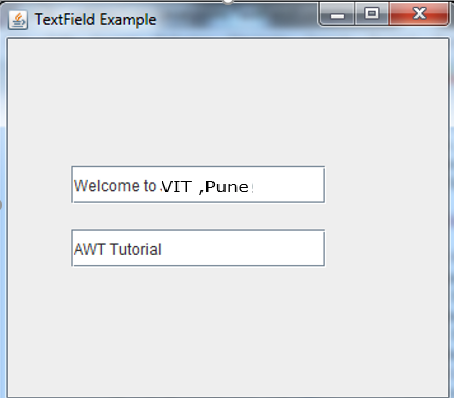
    f.setLayout(**null**);

    f.setVisible(**true**);

    }

    }

Output:



## **Java JTextField Example with ActionListener**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** TextFieldExample **implements** ActionListener{

    JTextField tf1,tf2,tf3;

    JButton b1,b2;

    TextFieldExample(){

        JFrame f= **new** JFrame();

        tf1=**new** JTextField();

        tf1.setBounds(50,50,150,20);

        tf2=**new** JTextField();

        tf2.setBounds(50,100,150,20);

        tf3=**new** JTextField();

        tf3.setBounds(50,150,150,20);

        tf3.setEditable(**false**);

        b1=**new** JButton("+");

        b1.setBounds(50,200,50,50);

        b2=**new** JButton("-");

        b2.setBounds(120,200,50,50);

        b1.addActionListener(**this**);

        b2.addActionListener(**this**);

        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);

        f.setSize(300,300);

        f.setLayout(**null**);

        f.setVisible(**true**);

    }

**public** **void** actionPerformed(ActionEvent e) {

        String s1=tf1.getText();

        String s2=tf2.getText();

**int** a=Integer.parseInt(s1);

**int** b=Integer.parseInt(s2);

**int** c=0;

**if**(e.getSource()==b1){

            c=a+b;

        }**else** **if**(e.getSource()==b2){

            c=a-b;

        }

        String result=String.valueOf(c);

        tf3.setText(result);

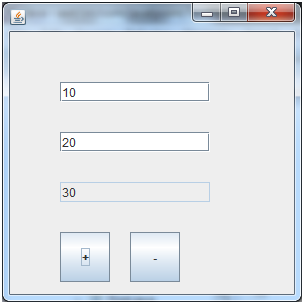
    }

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

**new** TextFieldExample();

} }

Output:



# Java 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. It inherits JTextField class.

## **JPasswordField class declaration**

Let's see the declaration for javax.swing.JPasswordField class.

**public** **class** JPasswordField **extends** JTextField

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JPasswordField() | Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width. |
| JPasswordField(int columns) | Constructs a new empty JPasswordField with the specified number of columns. |
| JPasswordField(String text) | Constructs a new JPasswordField initialized with the specified text. |
| JPasswordField(String text, int columns) | Construct a new JPasswordField initialized with the specified text and columns. |

## **Java JPasswordField Example**

**import** javax.swing.\*;

**public** **class** PasswordFieldExample {

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

    JFrame f=**new** JFrame("Password Field Example");

     JPasswordField value = **new** JPasswordField();

     JLabel l1=**new** JLabel("Password:");

        l1.setBounds(20,100, 80,30);

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

            f.add(value);  f.add(l1);

            f.setSize(300,300);

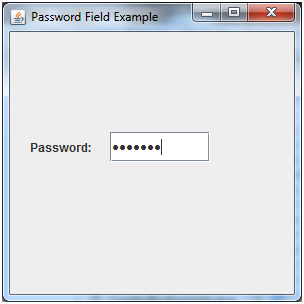
            f.setLayout(**null**);

            f.setVisible(**true**);

}

}

Output:



## **Java JPasswordField Example with ActionListener**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** PasswordFieldExample {

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

    JFrame f=**new** JFrame("Password Field Example");

**final** JLabel label = **new** JLabel();

     label.setBounds(20,150, 200,50);

**final** JPasswordField value = **new** JPasswordField();

     value.setBounds(100,75,100,30);

     JLabel l1=**new** JLabel("Username:");

        l1.setBounds(20,20, 80,30);

        JLabel l2=**new** JLabel("Password:");

        l2.setBounds(20,75, 80,30);

        JButton b = **new** JButton("Login");

        b.setBounds(100,120, 80,30);

**final** JTextField text = **new** JTextField();

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

                f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);

                f.setSize(300,300);

                f.setLayout(**null**);

                f.setVisible(**true**);

                b.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

                   String data = "Username " + text.getText();

                   data += ", Password: "

                   + **new** String(value.getPassword());

                   label.setText(data);

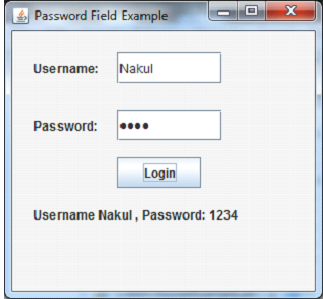
                }

             });

}

}

Output:



# Java JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits [JToggleButton](https://www.javatpoint.com/java-jtogglebutton) class.

## **JCheckBox class declaration**

Let's see the declaration for javax.swing.JCheckBox class.

**public** **class** JCheckBox **extends** JToggleButton **implements** Accessible

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JJCheckBox() | Creates an initially unselected check box button with no text, no icon. |
| JChechBox(String s) | Creates an initially unselected check box with text. |
| JCheckBox(String text, boolean selected) | Creates a check box with text and specifies whether or not it is initially selected. |
| JCheckBox(Action a) | Creates a check box where properties are taken from the Action supplied. |

### **Commonly used Methods:**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| AccessibleContext getAccessibleContext() | It is used to get the AccessibleContext associated with this JCheckBox. |
| protected String paramString() | It returns a [string](https://www.javatpoint.com/java-string) representation of this JCheckBox. |

## **Java JCheckBox Example**

**import** javax.swing.\*;

**public** **class** CheckBoxExample

{

     CheckBoxExample(){

        JFrame f= **new** JFrame("CheckBox Example");

        JCheckBox checkBox1 = **new** JCheckBox("C++");

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

        JCheckBox checkBox2 = **new** JCheckBox("Java", **true**);

        checkBox2.setBounds(100,150, 50,50);

        f.add(checkBox1);

        f.add(checkBox2);

        f.setSize(400,400);

        f.setLayout(**null**);

        f.setVisible(**true**);

     }

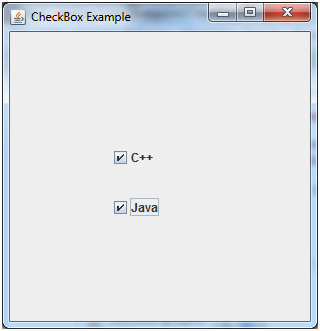
**public** **static** **void** main(String args[])

    {

**new** CheckBoxExample();

    }}

Output:



## **Java JCheckBox Example with ItemListener**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** CheckBoxExample

{

     CheckBoxExample(){

        JFrame f= **new** JFrame("CheckBox Example");

**final** JLabel label = **new** JLabel();

        label.setHorizontalAlignment(JLabel.CENTER);

        label.setSize(400,100);

        JCheckBox checkbox1 = **new** JCheckBox("C++");

        checkbox1.setBounds(150,100, 50,50);

        JCheckBox checkbox2 = **new** JCheckBox("Java");

        checkbox2.setBounds(150,150, 50,50);

        f.add(checkbox1); f.add(checkbox2); f.add(label);

        checkbox1.addItemListener(**new** ItemListener() {

**public** **void** itemStateChanged(ItemEvent e) {

                label.setText("C++ Checkbox: "

                + (e.getStateChange()==1?"checked":"unchecked"));

             }

          });

        checkbox2.addItemListener(**new** ItemListener() {

**public** **void** itemStateChanged(ItemEvent e) {

                label.setText("Java Checkbox: "

                + (e.getStateChange()==1?"checked":"unchecked"));

             }

          });

        f.setSize(400,400);

        f.setLayout(**null**);

        f.setVisible(**true**);

     }

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

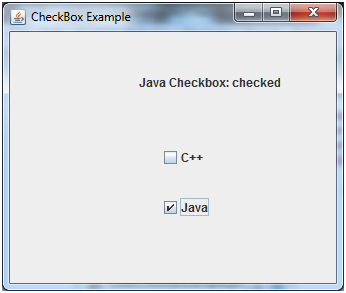
{

**new** CheckBoxExample();

}

}

Output:



## **Java JCheckBox Example: Food Order**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** CheckBoxExample **extends** JFrame **implements** ActionListener{

    JLabel l;

    JCheckBox cb1,cb2,cb3;

    JButton b;

    CheckBoxExample(){

        l=**new** JLabel("Food Ordering System");

        l.setBounds(50,50,300,20);

        cb1=**new** JCheckBox("Pizza @ 100");

        cb1.setBounds(100,100,150,20);

        cb2=**new** JCheckBox("Burger @ 30");

        cb2.setBounds(100,150,150,20);

        cb3=**new** JCheckBox("Tea @ 10");

        cb3.setBounds(100,200,150,20);

        b=**new** JButton("Order");

        b.setBounds(100,250,80,30);

        b.addActionListener(**this**);

        add(l);add(cb1);add(cb2);add(cb3);add(b);

        setSize(400,400);

        setLayout(**null**);

        setVisible(**true**);

        setDefaultCloseOperation(EXIT\_ON\_CLOSE);

    }

**public** **void** actionPerformed(ActionEvent e){

**float** amount=0;

        String msg="";

**if**(cb1.isSelected()){

            amount+=100;

            msg="Pizza: 100\n";

        }

**if**(cb2.isSelected()){

            amount+=30;

            msg+="Burger: 30\n";

        }

**if**(cb3.isSelected()){

            amount+=10;

            msg+="Tea: 10\n";

        }

        msg+="-----------------\n";

        JOptionPane.showMessageDialog(**this**,msg+"Total: "+amount);

    }

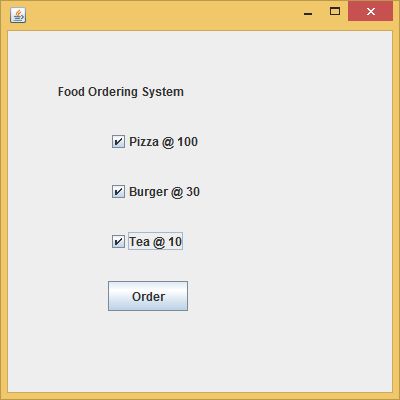
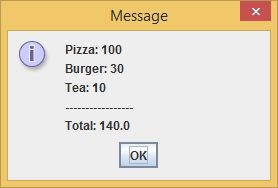
**public** **static** **void** main(String[] args) {

**new** CheckBoxExample();

    }

}

Output:

# Java 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.

## **JRadioButton class declaration**

Let's see the declaration for javax.swing.JRadioButton class.

**public** **class** JRadioButton **extends** JToggleButton **implements** Accessible

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JRadioButton() | Creates an unselected radio button with no text. |
| JRadioButton(String s) | Creates an unselected radio button with specified text. |
| JRadioButton(String s, boolean selected) | Creates a radio button with the specified text and selected status. |

### **Commonly used Methods:**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void setText(String s) | It is used to set specified text on button. |
| String getText() | It is used to return the text of the button. |
| void setEnabled(boolean b) | It is used to enable or disable the button. |
| void setIcon(Icon b) | It is used to set the specified Icon on the button. |
| Icon getIcon() | It is used to get the Icon of the button. |
| void setMnemonic(int a) | It is used to set the mnemonic on the button. |
| void addActionListener(ActionListener a) | It is used to add the action listener to this object. |

## **Java JRadioButton Example**

**import** javax.swing.\*;

**public** **class** RadioButtonExample {

JFrame f;

RadioButtonExample(){

f=**new** JFrame();

JRadioButton r1=**new** JRadioButton("A) Male");

JRadioButton r2=**new** JRadioButton("B) Female");

r1.setBounds(75,50,100,30);

r2.setBounds(75,100,100,30);

ButtonGroup bg=**new** ButtonGroup();

bg.add(r1);bg.add(r2);

f.add(r1);f.add(r2);

f.setSize(300,300);

f.setLayout(**null**);

f.setVisible(**true**);

}

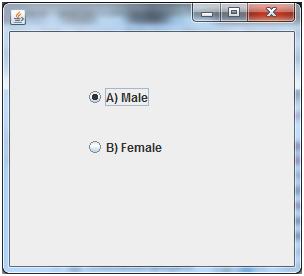
**public** **static** **void** main(String[] args) {

**new** RadioButtonExample();

}

}

Output:



## **Java JRadioButton Example with ActionListener**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**class** RadioButtonExample **extends** JFrame **implements** ActionListener{

JRadioButton rb1,rb2;

JButton b;

RadioButtonExample(){

rb1=**new** JRadioButton("Male");

rb1.setBounds(100,50,100,30);

rb2=**new** JRadioButton("Female");

rb2.setBounds(100,100,100,30);

ButtonGroup bg=**new** ButtonGroup();

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

b=**new** JButton("click");

b.setBounds(100,150,80,30);

b.addActionListener(**this**);

add(rb1);add(rb2);add(b);

setSize(300,300);

setLayout(**null**);

setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e){

**if**(rb1.isSelected()){

JOptionPane.showMessageDialog(**this**,"You are Male.");

}

**if**(rb2.isSelected()){

JOptionPane.showMessageDialog(**this**,"You are Female.");

}

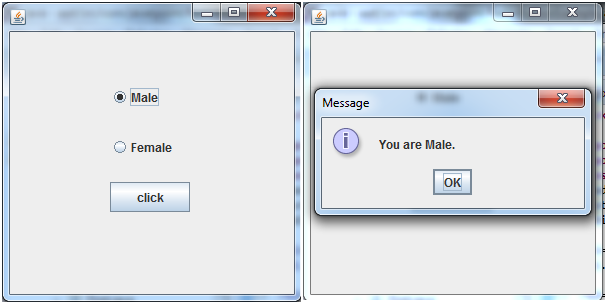
}

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

**new** RadioButtonExample();

}}

Output:



# Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a [menu](https://www.javatpoint.com/java-jmenuitem-and-jmenu). It inherits [JComponent](https://www.javatpoint.com/java-jcomponent) class.

## **JComboBox class declaration**

Let's see the declaration for javax.swing.JComboBox class.

**public** **class** JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JComboBox() | Creates a JComboBox with a default data model. |
| JComboBox(Object[] items) | Creates a JComboBox that contains the elements in the specified [array](https://www.javatpoint.com/array-in-java). |
| JComboBox(Vector<?> items) | Creates a JComboBox that contains the elements in the specified [Vector](https://www.javatpoint.com/scala-vector). |

### **Commonly used Methods:**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void addItem(Object anObject) | It is used to add an item to the item list. |
| void removeItem(Object anObject) | It is used to delete an item to the item list. |
| void removeAllItems() | It is used to remove all the items from the list. |
| void setEditable(boolean b) | It is used to determine whether the JComboBox is editable. |
| void addActionListener(ActionListener a) | It is used to add the [ActionListener](https://www.javatpoint.com/java-actionlistener). |
| void addItemListener(ItemListener i) | It is used to add the [ItemListener](https://www.javatpoint.com/java-itemlistener). |

## **Java JComboBox Example**

**import** javax.swing.\*;

**public** **class** ComboBoxExample {

JFrame f;

ComboBoxExample(){

    f=**new** JFrame("ComboBox Example");

    String country[]={"India","Aus","U.S.A","England","Newzealand"};

    JComboBox cb=**new** JComboBox(country);

    cb.setBounds(50, 50,90,20);

    f.add(cb);

    f.setLayout(**null**);

    f.setSize(400,500);

    f.setVisible(**true**);

}

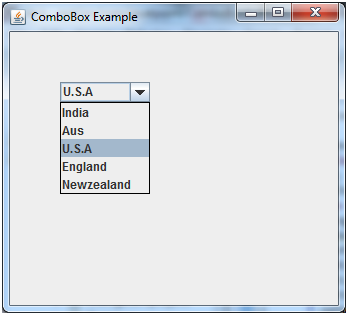
**public** **static** **void** main(String[] args) {

**new** ComboBoxExample();

}

}

Output:



## **Java JComboBox Example with ActionListener**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** ComboBoxExample {

JFrame f;

ComboBoxExample(){

    f=**new** JFrame("ComboBox Example");

**final** JLabel label = **new** JLabel();

    label.setHorizontalAlignment(JLabel.CENTER);

    label.setSize(400,100);

    JButton b=**new** JButton("Show");

    b.setBounds(200,100,75,20);

    String languages[]={"C","C++","C#","Java","PHP"};

**final** JComboBox cb=**new** JComboBox(languages);

    cb.setBounds(50, 100,90,20);

    f.add(cb); f.add(label); f.add(b);

    f.setLayout(**null**);

    f.setSize(350,350);

    f.setVisible(**true**);

    b.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String data = "Programming language Selected: "

   + cb.getItemAt(cb.getSelectedIndex());

label.setText(data);

}

});

}

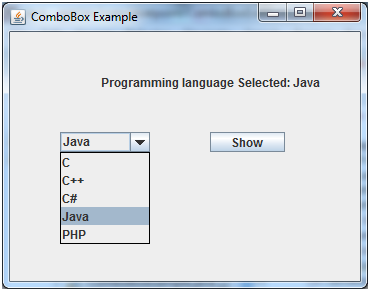
**public** **static** **void** main(String[] args) {

**new** ComboBoxExample();

}

}

Output:



# Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

## **JTable class declaration**

Let's see the declaration for javax.swing.JTable class.

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JTable() | Creates a table with empty cells. |
| JTable(Object[][] rows, Object[] columns) | Creates a table with the specified data. |

## **Java JTable Example**

**import** javax.swing.\*;

**public** **class** TableExample {

    JFrame f;

    TableExample(){

    f=**new** JFrame();

    String data[][]={ {"101","Amit","670000"},

                          {"102","Jai","780000"},

                          {"101","Sachin","700000"}};

    String column[]={"ID","NAME","SALARY"};

    JTable jt=**new** JTable(data,column);

    jt.setBounds(30,40,200,300);

    JScrollPane sp=**new** JScrollPane(jt);

    f.add(sp);

    f.setSize(300,400);

    f.setVisible(**true**);

}

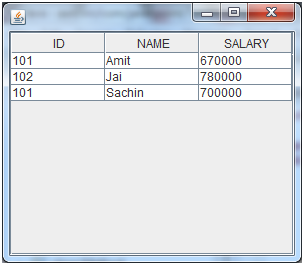
**public** **static** **void** main(String[] args) {

**new** TableExample();

}

}

Output:



## **Java JTable Example with ListSelectionListener**

**import** javax.swing.\*;

**import** javax.swing.event.\*;

**public** **class** TableExample {

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

            JFrame f = **new** JFrame("Table Example");

            String data[][]={ {"101","Amit","670000"},

                                                                       {"102","Jai","780000"},

                                                                       {"101","Sachin","700000"}};

                            String column[]={"ID","NAME","SALARY"};

**final** JTable jt=**new** JTable(data,column);

            jt.setCellSelectionEnabled(**true**);

            ListSelectionModel select= jt.getSelectionModel();

            select.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

            select.addListSelectionListener(**new** ListSelectionListener() {

**public** **void** valueChanged(ListSelectionEvent e) {

                String Data = **null**;

**int**[] row = jt.getSelectedRows();

**int**[] columns = jt.getSelectedColumns();

**for** (**int** i = 0; i < row.length; i++) {

**for** (**int** j = 0; j < columns.length; j++) {

                    Data = (String) jt.getValueAt(row[i], columns[j]);

                  } }

                System.out.println("Table element selected is: " + Data);

              }

            });

            JScrollPane sp=**new** JScrollPane(jt);

            f.add(sp);

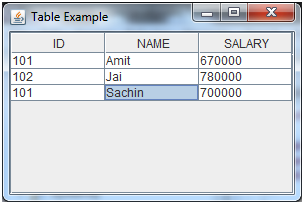
            f.setSize(300, 200);

            f.setVisible(**true**);

          }

        }

Output:



If you select an element in column **NAME**, name of the element will be displayed on the console: Table element selected is: Sachin

# Java JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

## **JList class declaration**

Let's see the declaration for javax.swing.JList class.

**public** **class** JList **extends** JComponent **implements** Scrollable, Accessible

### **Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JList() | Creates a JList with an empty, read-only, model. |
| JList(ary[] listData) | Creates a JList that displays the elements in the specified array. |
| JList(ListModel<ary> dataModel) | Creates a JList that displays elements from the specified, non-null, model. |

### **Commonly used Methods:**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Void addListSelectionListener(ListSelectionListener listener) | It is used to add a listener to the list, to be notified each time a change to the selection occurs. |
| int getSelectedIndex() | It is used to return the smallest selected cell index. |
| ListModel getModel() | It is used to return the data model that holds a list of items displayed by the JList component. |
| void setListData(Object[] listData) | It is used to create a read-only ListModel from an array of objects. |

## **Java JList Example**

**import** javax.swing.\*;

**public** **class** ListExample

{

     ListExample(){

        JFrame f= **new** JFrame();

        DefaultListModel<String> l1 = **new** DefaultListModel<>();

          l1.addElement("Item1");

          l1.addElement("Item2");

          l1.addElement("Item3");

          l1.addElement("Item4");

          JList<String> list = **new** JList<>(l1);

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

          f.add(list);

          f.setSize(400,400);

          f.setLayout(**null**);

          f.setVisible(**true**);

     }

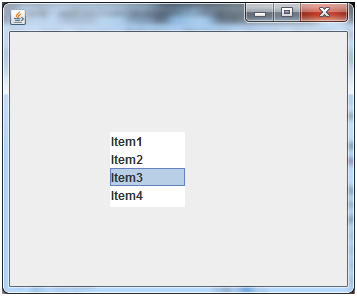
**public** **static** **void** main(String args[])

    {

**new** ListExample();

    }}

Output:



### **Java JList Example with ActionListener**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** ListExample

{

     ListExample(){

        JFrame f= **new** JFrame();

**final** JLabel label = **new** JLabel();

        label.setSize(500,100);

        JButton b=**new** JButton("Show");

        b.setBounds(200,150,80,30);

**final** DefaultListModel<String> l1 = **new** DefaultListModel<>();

          l1.addElement("C");

          l1.addElement("C++");

          l1.addElement("Java");

          l1.addElement("PHP");

**final** JList<String> list1 = **new** JList<>(l1);

          list1.setBounds(100,100, 75,75);

          DefaultListModel<String> l2 = **new** DefaultListModel<>();

          l2.addElement("Turbo C++");

          l2.addElement("Struts");

          l2.addElement("Spring");

          l2.addElement("YII");

**final** JList<String> list2 = **new** JList<>(l2);

          list2.setBounds(100,200, 75,75);

          f.add(list1); f.add(list2); f.add(b); f.add(label);

          f.setSize(450,450);

          f.setLayout(**null**);

          f.setVisible(**true**);

          b.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

                 String data = "";

**if** (list1.getSelectedIndex() != -1) {

                    data = "Programming language Selected: " + list1.getSelectedValue();

                    label.setText(data);

                 }

**if**(list2.getSelectedIndex() != -1){

                    data += ", FrameWork Selected: ";

**for**(Object frame :list2.getSelectedValues()){

                       data += frame + " ";

                    }

                 }

                 label.setText(data);

              }

           });

     }

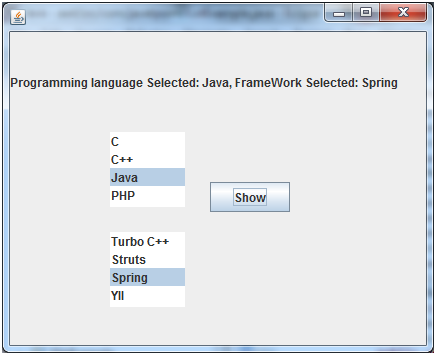
**public** **static** **void** main(String args[])

    {

**new** ListExample();

    }}

Output:



# 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.

### **Nested Class**

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Class** | **Description** |
| protected class | JFrame.AccessibleJFrame | This class implements accessibility support for the JFrame class. |

### **Fields**

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Field** | **Description** |
| protected AccessibleContext | accessibleContext | The accessible context property. |
| static int | EXIT\_ON\_CLOSE | The exit application default window close operation. |
| protected JRootPane | rootPane | The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane. |
| protected boolean | rootPaneCheckingEnabled | If true then calls to add and setLayout will be forwarded to the contentPane. |

### **Constructors**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JFrame() | It constructs a new frame that is initially invisible. |
| JFrame(GraphicsConfiguration gc) | It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title. |
| JFrame(String title) | It creates a new, initially invisible Frame with the specified title. |
| JFrame(String title, GraphicsConfiguration gc) | It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device. |

### **Useful Methods**

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| protected void | addImpl(Component comp, Object constraints, int index) | Adds the specified child Component. |
| protected JRootPane | createRootPane() | Called by the constructor methods to create the default rootPane. |
| protected void | frameInit() | Called by the constructors to init the JFrame properly. |
| void | setContentPane(Containe contentPane) | It sets the contentPane property |
| static void | setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated) | Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel. |
| void | setIconImage(Image image) | It sets the image to be displayed as the icon for this window. |
| void | setJMenuBar(JMenuBar menubar) | It sets the menubar for this frame. |
| void | setLayeredPane(JLayeredPane layeredPane) | It sets the layeredPane property. |
| JRootPane | getRootPane() | It returns the rootPane object for this frame. |
| TransferHandler | getTransferHandler() | It gets the transferHandler property. |

## **JFrame Example**

**import** java.awt.FlowLayout;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.Jpanel;

**public** **class** JFrameExample {

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

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

        JPanel panel = **new** JPanel();

        panel.setLayout(**new** FlowLayout());

        JLabel label = **new** JLabel("JFrame By Example");

        JButton button = **new** JButton();

        button.setText("Button");

        panel.add(label);

        panel.add(button);

        frame.add(panel);

        frame.setSize(200, 300);

        frame.setLocationRelativeTo(**null**);

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

        frame.setVisible(**true**);

    }

}

Output



# Java JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

### **JMenuBar class declaration**

**public** **class** JMenuBar **extends** JComponent **implements** MenuElement, Accessible

### **JMenu class declaration**

**public** **class** JMenu **extends** JMenuItem **implements** MenuElement, Accessible

### **JMenuItem class declaration**

**public** **class** JMenuItem **extends** AbstractButton **implements** Accessible, MenuElement

## **Java JMenuItem and JMenu Example**

**import** javax.swing.\*;

**class** MenuExample

{

          JMenu menu, submenu;

          JMenuItem i1, i2, i3, i4, i5;

          MenuExample(){

          JFrame f= **new** JFrame("Menu and MenuItem Example");

          JMenuBar mb=**new** JMenuBar();

          menu=**new** JMenu("Menu");

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

          i1=**new** JMenuItem("Item 1");

          i2=**new** JMenuItem("Item 2");

          i3=**new** JMenuItem("Item 3");

          i4=**new** JMenuItem("Item 4");

          i5=**new** JMenuItem("Item 5");

          menu.add(i1); menu.add(i2); menu.add(i3);

          submenu.add(i4); submenu.add(i5);

          menu.add(submenu);

          mb.add(menu);

          f.setJMenuBar(mb);

          f.setSize(400,400);

          f.setLayout(**null**);

          f.setVisible(**true**);

}

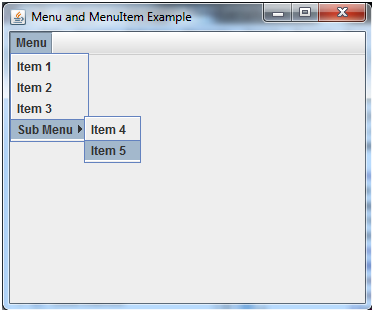
**public** **static** **void** main(String args[])

{

**new** MenuExample();

}}

Output:



## **Example of creating Edit menu for Notepad:**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** MenuExample **implements** ActionListener{

JFrame f;

JMenuBar mb;

JMenu file,edit,help;

JMenuItem cut,copy,paste,selectAll;

JTextArea ta;

MenuExample(){

f=**new** JFrame();

cut=**new** JMenuItem("cut");

copy=**new** JMenuItem("copy");

paste=**new** JMenuItem("paste");

selectAll=**new** JMenuItem("selectAll");

cut.addActionListener(**this**);

copy.addActionListener(**this**);

paste.addActionListener(**this**);

selectAll.addActionListener(**this**);

mb=**new** JMenuBar();

file=**new** JMenu("File");

edit=**new** JMenu("Edit");

help=**new** JMenu("Help");

edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);

mb.add(file);mb.add(edit);mb.add(help);

ta=**new** JTextArea();

ta.setBounds(5,5,360,320);

f.add(mb);f.add(ta);

f.setJMenuBar(mb);

f.setLayout(**null**);

f.setSize(400,400);

f.setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getSource()==cut)

ta.cut();

**if**(e.getSource()==paste)

ta.paste();

**if**(e.getSource()==copy)

ta.copy();

**if**(e.getSource()==selectAll)

ta.selectAll();

}

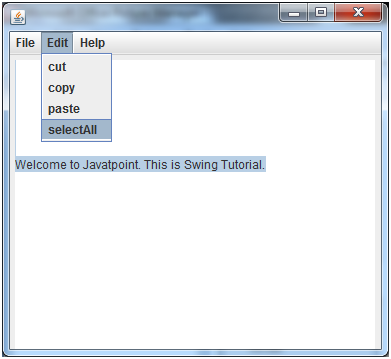
**public** **static** **void** main(String[] args) {

**new** MenuExample();

}

}

Output:



## **Java JPopupMenu Example with MouseListener and ActionListener**

**import** javax.swing.\*;

**import** java.awt.event.\*;

**class** PopupMenuExample

{

     PopupMenuExample(){

**final** JFrame f= **new** JFrame("PopupMenu Example");

**final** JLabel label = **new** JLabel();

         label.setHorizontalAlignment(JLabel.CENTER);

         label.setSize(400,100);

**final** 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());

            }

         });

        cut.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

             label.setText("cut MenuItem clicked.");

         }

        });

        copy.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

                label.setText("copy MenuItem clicked.");

            }

           });

        paste.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent e) {

                label.setText("paste MenuItem clicked.");

            }

           });

         f.add(label); f.add(popupmenu);

         f.setSize(400,400);

         f.setLayout(**null**);

         f.setVisible(**true**);

     }

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

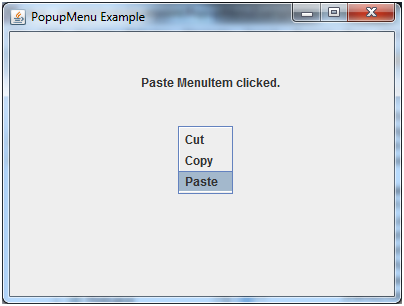
{

**new** PopupMenuExample();

}

}

Output:



# BorderLayout (LayoutManagers)

### **Java LayoutManagers**

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

### **Java BorderLayout**

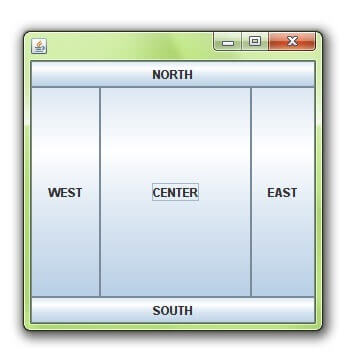
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

### **Constructors of BorderLayout class:**

* **BorderLayout():** creates a border layout but with no gaps between the components.
* **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

### **Example of BorderLayout class:**



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** Border {

JFrame f;

Border(){

    f=**new** JFrame();

    JButton b1=**new** JButton("NORTH");;

    JButton b2=**new** JButton("SOUTH");;

    JButton b3=**new** JButton("EAST");;

    JButton b4=**new** JButton("WEST");;

    JButton b5=**new** JButton("CENTER");;

    f.add(b1,BorderLayout.NORTH);

    f.add(b2,BorderLayout.SOUTH);

    f.add(b3,BorderLayout.EAST);

    f.add(b4,BorderLayout.WEST);

    f.add(b5,BorderLayout.CENTER);

    f.setSize(300,300);

    f.setVisible(**true**);

}

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

**new** Border();

}

}

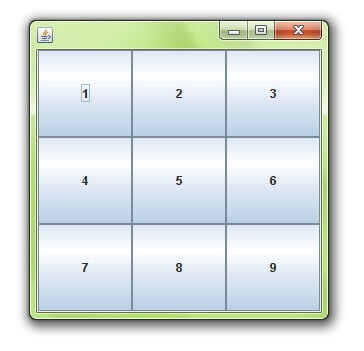
# Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

### **Constructors of GridLayout class**

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

### **Example of GridLayout class**



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** MyGridLayout{

JFrame f;

MyGridLayout(){

    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");

        JButton b6=**new** JButton("6");

        JButton b7=**new** JButton("7");

    JButton b8=**new** JButton("8");

        JButton b9=**new** JButton("9");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.add(b6);f.add(b7);f.add(b8);f.add(b9);

    f.setLayout(**new** GridLayout(3,3));

    //setting grid layout of 3 rows and 3 columns

    f.setSize(300,300);

    f.setVisible(**true**);

}

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

**new** MyGridLayout();

}

}

# Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

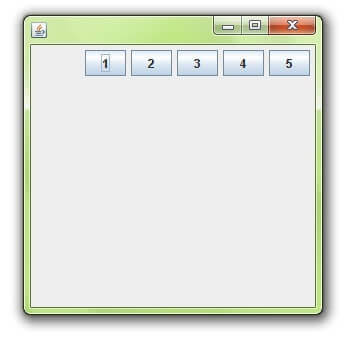
### **Fields of FlowLayout class**

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

### **Constructors of FlowLayout class**

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

### **Example of FlowLayout class**



**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");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.setLayout(**new** FlowLayout(FlowLayout.RIGHT));

    //setting flow layout of right alignment

    f.setSize(300,300);

    f.setVisible(**true**);

}

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

**new** MyFlowLayout();

}

}

# Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

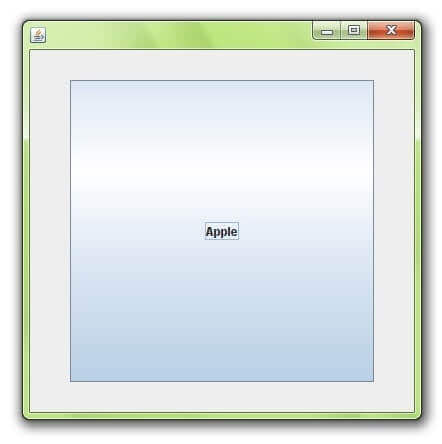
### **Constructors of CardLayout class**

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

### **Commonly used methods of CardLayout class**

* **public void next(Container parent):** is used to flip to the next card of the given container.
* **public void previous(Container parent):** is used to flip to the previous card of the given container.
* **public void first(Container parent):** is used to flip to the first card of the given container.
* **public void last(Container parent):** is used to flip to the last card of the given container.
* **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

### **Example of CardLayout class**



**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** CardLayoutExample **extends** JFrame **implements** ActionListener{

CardLayout card;

JButton b1,b2,b3;

Container c;

    CardLayoutExample(){

        c=getContentPane();

        card=**new** CardLayout(40,30);

//create CardLayout object with 40 hor space and 30 ver space

        c.setLayout(card);

        b1=**new** JButton("Apple");

        b2=**new** JButton("Boy");

        b3=**new** JButton("Cat");

        b1.addActionListener(**this**);

        b2.addActionListener(**this**);

        b3.addActionListener(**this**);

        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }

**public** **void** actionPerformed(ActionEvent e) {

    card.next(c);

    }

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

        CardLayoutExample cl=**new** CardLayoutExample();

        cl.setSize(400,400);

        cl.setVisible(**true**);

        cl.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

    }

}

USEFUL LINKS

<https://www.clear.rice.edu/comp310/JavaResources/GUI/>

https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a\_gui.html