**Java Applet**

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

### Advantage of Applet

There are many advantages of applet. They are as follows:

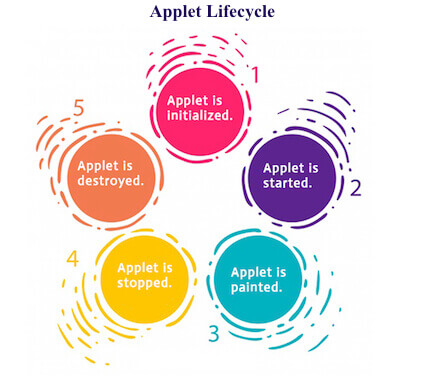
* It works at client side so less response time.
* Secured
* It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

### Drawback of Applet

* Plugin is required at client browser to execute applet.

### Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



### Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

### java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

### java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

### How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

### Simple example

1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
6. public void paint(Graphics g){
7. g.drawString("welcome to applet",150,150);
8. }
10. }
11. /\*
12. <applet code="First.class" width="300" height="300">
13. </applet>
14. \*/
15. To execute the applet by appletviewer tool, write in command prompt:
16. **c:\>**javac First.java
17. **c:\>**appletviewer First.java

# Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

## Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

## Example of Graphics in applet:

import java.applet.Applet;

import java.awt.\*;

public class GraphicsDemo extends Applet{

public void paint(Graphics g){

g.setColor(Color.red);

g.drawString("Welcome",50, 50);

g.drawLine(20,30,20,300);

g.drawRect(70,100,30,30);

g.fillRect(170,100,30,30);

g.drawOval(70,200,30,30);

g.setColor(Color.pink);

g.fillOval(170,200,30,30);

g.drawArc(90,150,30,30,30,270);

g.fillArc(270,150,30,30,0,180);

}

}

/\*

<applet code="GraphicsDemo.class" width="400" height="400">

</applet>

\*/

## JAVA AWT (Abstract Window Toolkit)

The Java AWT creates components by calling the subroutines of native platforms. Hence, an AWT GUI application will have the look and feel of Windows OS while running on Windows and Mac OS look and feel when running on Mac and so on. This explains the platform dependency of Abstract Window Toolkit applications.

Due to its platform-dependence and a kind of heavyweight nature of its components, it is rarely used in Java applications these days. Besides, there are also newer frameworks like Swing which are light-weight and platform-independent.

Swing has more flexible and powerful components when compared to AWT. Swing provides components similar to Abstract Window Toolkit and also has more advanced components like trees, tabbed panels, etc.

**A frame can be created in two ways:**

**#1) By using the Frame class object**

Here, we create a Frame class object by instantiating the Frame class.

**A programming example is given below.**

|  |
| --- |
| import java.awt.\*;      class FrameButton{          FrameButton (){              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[]){          FrameButton f=new FrameButton ();      }  } |

**#2) By Extending the Frame class**

Here we create a class that extends the Frame class and then create components of the frame in its constructor.

**This is shown in the program below.**

|  |
| --- |
| import java.awt.\*;  class AWTButton extends Frame{      AWTButton (){      Button b=new Button("AWTButton");      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[]){          AWTButton f=new AWTButton ();      }  } |

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.

|  |  |  |
| --- | --- | --- |
| 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**. |

|  |  |
| --- | --- |
| **Method (Swing)** | **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. |

**Layouts:**

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

### Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**

### Constructors of FlowLayout class

**FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

**FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

**FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

**Commonly used methods:**

1. **setTitle(String Text)**: This Method is used to set Title of JFrame. The title you want to set is passed as a string.
2. **setVisible(Bool)** This Method is used to set visibility.
3. setBounds(left,top,height,width) Size of layout.

**// Java program to show Example of FlowLayout.**

**// in java. Importing different Package.**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**class Example extends JFrame {**

**// Declaration of objects of JLabel class.**

**JLabel l1, l2, l3, l4, l5;**

**// Constructor of Example class.**

**public Example()**

**{**

**// Creating Object of "FlowLayout" class**

**FlowLayout layout = new FlowLayout();**

**// this Keyword refers to current object.**

**// Function to set Layout of JFrame.**

**this.setLayout(layout);**

**// Initialization of object "l1" of JLabel class.**

**l1 = new JLabel("Label 1 ");**

**// Initialization of object "l2" of JLabel class.**

**l2 = new JLabel("Label 2 ");**

**// Initialization of object "l3" of JLabel class.**

**l3 = new JLabel("Label 3 ");**

**// Initialization of object "l4" of JLabel class.**

**l4 = new JLabel("Label 4 ");**

**// Initialization of object "l5" of JLabel class.**

**l5 = new JLabel("Label 5 ");**

**// this Keyword refers to current object.**

**// Adding Jlabel "l1" on JFrame.**

**this.add(l1);**

**// Adding Jlabel "l2" on JFrame.**

**this.add(l2);**

**// Adding Jlabel "l3" on JFrame.**

**this.add(l3);**

**// Adding Jlabel "l4" on JFrame.**

**this.add(l4);**

**// Adding Jlabel "l5" on JFrame.**

**this.add(l5);**

**}**

**}**

**class MainFrame {**

**// Driver code**

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

**{**

**// Creating Object of Example class.**

**Example f = new Example();**

**// Function to set title of JFrame.**

**f.setTitle("Example of FlowLayout");**

**// Function to set Bounds of JFrame.**

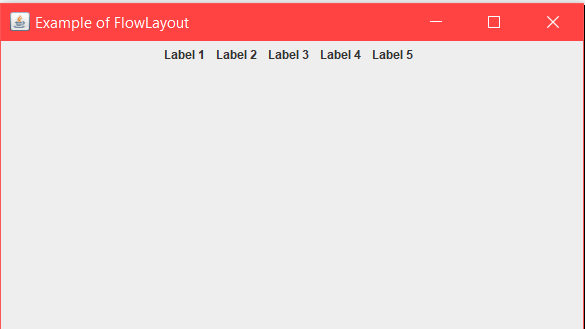
**f.setBounds(200, 100, 600, 400);**

**// Function to set visible status of JFrame.**

**f.setVisible(true);**

**}**

**}**



We can **control the alignment of components** in a flow layout arrangement, by using these FlowLayout Fields.

// Java program to show example of

// FlowLayout and using RIGHT alignment

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

class Example extends JFrame {

// Declaration of objects of JLabel class.

JLabel l1, l2, l3, l4, l5;

// Constructor of Example class.

public Example()

{

// Creating Object of "FlowLayout" class, passing

// RIGHT alignment through constructor.

FlowLayout layout = new FlowLayout(FlowLayout.RIGHT);

// this Keyword refers to current object.

// Function to set Layout of JFrame.

this.setLayout(layout);

// Initialization of object "l1" of JLabel class.

l1 = new JLabel("Label 1 ");

// Initialization of object "l2" of JLabel class.

l2 = new JLabel("Label 2 ");

// Initialization of object "l3" of JLabel class.

l3 = new JLabel("Label 3 ");

// Initialization of object "l4" of JLabel class.

l4 = new JLabel("Label 4 ");

// Initialization of object "l5" of JLabel class.

l5 = new JLabel("Label 5 ");

// this Keyword refers to current object.

// Adding Jlabel "l1" on JFrame.

this.add(l1);

// Adding Jlabel "l2" on JFrame.

this.add(l2);

// Adding Jlabel "l3" on JFrame.

this.add(l3);

// Adding Jlabel "l4" on JFrame.

this.add(l4);

// Adding Jlabel "l5" on JFrame.

this.add(l5);

}

}

class MainFrame {

// Driver code

public static void main(String[] args)

{

// Creating Object of Example class.

Example f = new Example();

// Function to set title of JFrame.

f.setTitle("Example of FlowLayout");

// Function to set Bounds of JFrame.

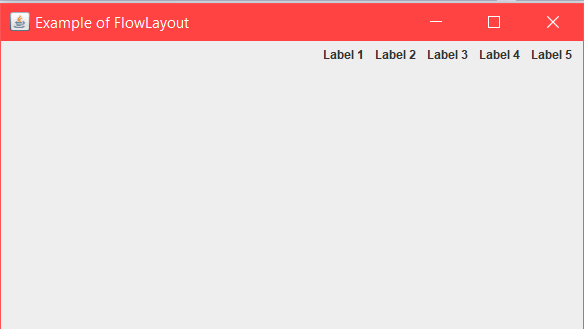
f.setBounds(200, 100, 600, 400);

// Function to set visible status of JFrame.

f.setVisible(true);

}

}



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

**Constructors:**

1. **BorderLayout():** It will construct a new borderlayout with no gaps between the components.
2. **BorderLayout(int, int):** It will constructs a border layout with the specified gaps between the components.

**Commonly Used Methods:**

1. **toString()**: Returns a string which is the representation of the state of border layout.
2. **getLayoutAlignmentX(Container parent)**: Returns the layout alignment along the X-axis.
3. **getLayoutAlignmentY(Container parent)**: It will return the layout alignment along the Y-axis.
4. **removeLayoutComponent(Component comp)**: This method is used to remove the specified component from the borderlayout.
5. **getVgap()**: Return the vertical gap between the components.
6. **getHgap()**: Returns the Horizontal gap between the components.
7. **setHgap(int hgap)**: It is used to set the horizontal gap between the components.
8. **setVgap(int vgap)**: It is used to set the vertical gap between the components.

// Java program to illustrate the BorderLayout

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

// class extends JFrame

class BoderLayoutDemo extends JFrame {

BoderLayoutDemo()

{

// Creating Object of Jpanel class

JPanel pa = new JPanel();

// set the layout

pa.setLayout(new BorderLayout());

// add a new JButton with name "wel" and it is

// lie top of the container

pa.add(new JButton("WelCome"), BorderLayout.NORTH);

// add a new JButton with name "come" and it is

// lie button of the container

pa.add(new JButton("KSC"), BorderLayout.SOUTH);

// add a new JButton with name "Layout" and it is

// lie left of the container

pa.add(new JButton("Layout"), BorderLayout.EAST);

// add a new JButton with name "Border" and it is

// lie right of the container

pa.add(new JButton("Border"), BorderLayout.WEST);

// add a new JButton with name "hello everybody" and it is

// lie center of the container

pa.add(new JButton("KSCPAC"), BorderLayout.CENTER);

// add the pa object which refer to the Jpanel

add(pa);

// Function to close the operation of JFrame.

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

// Function to set size of JFrame.

setSize(300, 300);

// Function to set visible status of JFrame.

setVisible(true);

}

}

class MainFrame {

// Driver code

public static void main(String[] args)

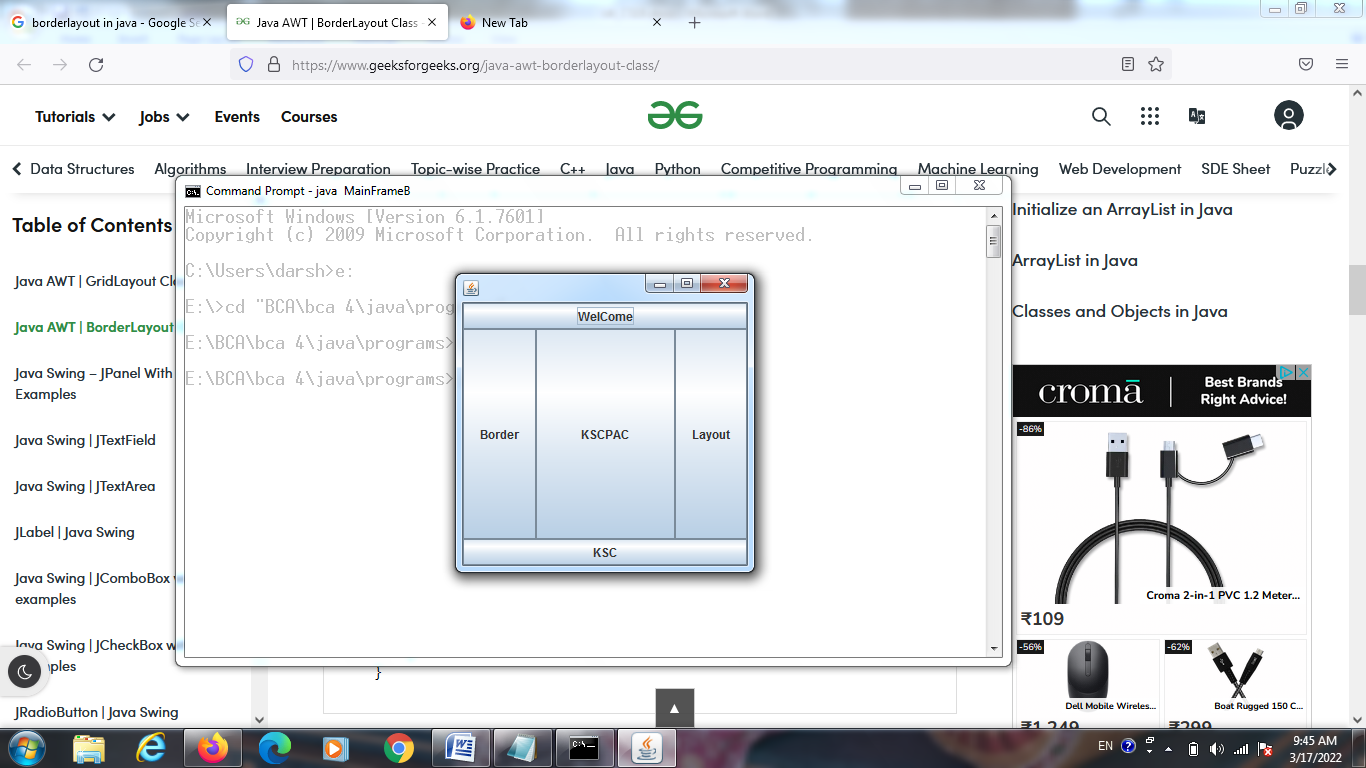
{

// calling the constructor

new BoderLayoutDemo();

}

}



* **CardLayout**: It arranges two or more components having the same size. The components are **arranged in a deck**, where all the cards of the same size and the**only top card are visible at any time**. The first component added in the container will be kept at the top of the deck. The default gap at the left, right, top and bottom edges are zero and the card components are displayed either **horizontally or vertically.**

**Constructors:**

1. **CardLayout():** It is used to create a new card layout with gaps of size is zero.
2. **CardLayout(int horizontalgap, int verticalgap):** It is used to create a new CardLayout class with the specified horizontal and vertical gaps.

|  |  |
| --- | --- |
| CardLayout (int, int) | Creates a new layout manager with the spacing horizontal and vertical specified. |
| first (Container) | Displays the first component added to the layout container specified |
| getHgap () | Gets the horizontal spacing. |
| getVgap () | Gets the vertical spacing. |
| last (Container) | Returns the last component added to the layout container specified |
| next (Container) | Returns the next component added to the layout container specified |
| previous (Container) | Displays the previous component added to the layout container specified |
| setHgap (int) | Specifies the horizontal spacing. |
| setVgap (int) | Specifies the vertical spacing. |

* package carddemo;
* import java.awt.\*;
* import java.awt.event.\*;
* class CardDemos extends Frame implements ActionListener
* {
* CardLayout card = new CardLayout(20,20);
* CardDemos()
* {
* setLayout(card);
* Button Btnfirst = new Button("first ");
* Button BtnSecond = new Button ("Second");
* Button BtnThird = new Button("Third");
* add(Btnfirst,"Card1");
* add(BtnSecond,"Card2");
* add(BtnThird,"Card3");
* Btnfirst.addActionListener(this);
* BtnSecond.addActionListener (this);
* BtnThird.addActionListener(this);
* }
* public void actionPerformed(ActionEvent e)
* {
* card.next(this);
* }
* }
* class CardDemo
* {
* public static void main(String args[])
* {
* CardDemos frame = new CardDemos();
* frame.setTitle("CardLayout in Java Example");
* frame.setSize(220,150);
* frame.setResizable(false);
* frame.setVisible(true);
* }
* }
* **GridLayout**: It arranges all the components in a grid of **equally sized cells**, adding them from the **left to righ**t and **top to bottom**. Only one component can be placed in a cell and each region of the grid will have the same size. When the container is resized, all cells are automatically resized.

### 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 along with given horizontal and vertical gaps.

// Java program to illustrate the GridLayout

import javax.swing.\*;

import java.awt.\*;

// class GridLayout extends JFrame

public class GridLayoutDemo extends JFrame {

GridLayoutDemo() {

// Creating Object P1 of JPanel class

JPanel p1 = new JPanel();

// set the layout

p1.setLayout(new GridLayout(4, 2));

// Creating Object of "FlowLayout" class

FlowLayout layout = new FlowLayout();

// Creating Object P2 of JPanel class

JPanel p2 = new JPanel();

// set the layout

p2.setLayout(layout);

// Declaration of objects of JLabel class.

JLabel one, two, three, four;

// Declaration of objects of JTextField class.

JTextField tname, tsalary, tcode, tdesig;

// Declaration of objects of JButton class.

JButton buttonSave, buttonExit;

// Initialization of object

// "one" of JLabel class.

one = new JLabel("NAME");

// Initialization of object

// "tname" of JTextField class.

tname = new JTextField(20);

// Initialization of object

// "two" of JLabel class.

two = new JLabel("CODE");

// Initialization of object

// "tcode" of JTextField class.

tcode = new JTextField(20);

// Initialization of object

// "three" of JLabel class.

three = new JLabel("DESIGNATION");

// Initialization of object

// "tdesig" of JTextField class.

tdesig = new JTextField(20);

// Initialization of object

// "four" of JLabel class.

four = new JLabel("SALARY");

// Initialization of object

// "tsalary" of JTextField class.

tsalary = new JTextField(20);

// Initialization of object

// "buttonsave" of JButton class.

buttonSave = new JButton("SAVE");

// Initialization of object

// "buttonexit" of JButton class.

buttonExit = new JButton("EXIT");

// Adding Jlabel "one" on JFrame.

p1.add(one);

// Adding JTextField "tname" on JFrame.

p1.add(tname);

// Adding Jlabel "two" on JFrame.

p1.add(two);

// Adding JTextField "tcode" on JFrame.

p1.add(tcode);

// Adding Jlabel "three" on JFrame.

p1.add(three);

// Adding JTextField "tdesig" on JFrame.

p1.add(tdesig);

// Adding Jlabel "four" on JFrame.

p1.add(four);

// Adding JTextField "tsalary" on JFrame.

p1.add(tsalary);

// Adding JButton "buttonsave" on JFrame.

p2.add(buttonSave);

// Adding JButton "buttonexit" on JFrame.

p2.add(buttonExit);

// add the p1 object which

// refer to the Jpanel

add(p1, "North");

// add the p2 object which

// refer to the Jpanel

add(p2, "South");

// Function to set visible

// status of JFrame.

setVisible(true);

// this Keyword refers to current

// object. Function to set size of JFrame.

this.setSize(400, 180);

}

// Main Method

public static void main(String args[])

{

// calling the constructor

new GridLayoutDemo();

}

}

* **BoxLayout**: It arranges multiple components in either **vertically or horizontally**, but not both. The components are arranged from**left to right or top to bottom**. If the components are aligned **horizontally**, the height of all components will be the same and equal to the largest sized components. If the components are aligned **vertically**, the width of all components will be the same and equal to the largest width components.

### Fields of BoxLayout Class

1. **public static final int X\_AXIS:** Alignment of the components are horizontal from left to right.
2. **public static final int Y\_AXIS:** Alignment of the components are vertical from top to bottom.
3. **public static final int LINE\_AXIS:** Alignment of the components is similar to the way words are aligned in a line, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then the components are aligned horizontally; otherwise, the components are aligned vertically. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is also from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.
4. **public static final int PAGE\_AXIS:** Alignment of the components is similar to the way text lines are put on a page, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then components are aligned vertically; otherwise, the components are aligned horizontally. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is also from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

### Constructor of BoxLayout class

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

### Example of BoxLayout class with Y-AXIS:

**FileName:** BoxLayoutExample1.java

import java.awt.\*;

import javax.swing.\*;

public class BoxLayoutExample1 extends Frame {

 Button buttons[];

 public BoxLayoutExample1 () {

   buttons = new Button [5];

   for (int i = 0;i<5;i++) {

      buttons[i] = new Button ("Button " + (i + 1));

      // adding the buttons so that it can be displayed

      add (buttons[i]);

    }

  // the buttons will be placed horizontally

setLayout (new BoxLayout (this, BoxLayout.Y\_AXIS));    //X\_AXIS

setSize(400,400);

setVisible(true);

}

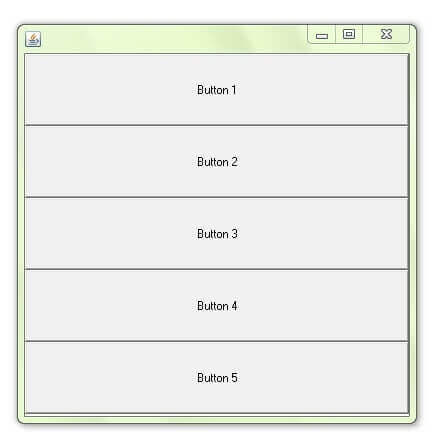
  // main method

public static void main(String args[]){

BoxLayoutExample1 b=new BoxLayoutExample1();

}

}



### Example of BoxLayout Class with LINE\_AXIS

The following example shows the effect of setting the value of ComponentOrientation property of the container to RIGHT\_TO\_LEFT. If we do not set the value of ComponentOrientation property, then the components would be laid out from left to right. Comment line 11, and see it yourself.

**FileName:** BoxLayoutExample3.java

// import statements

import java.awt.\*;

import javax.swing.\*;

public class BoxLayoutExample3 extends Frame

{

Button buttons[];

// constructor of the class

public BoxLayoutExample3()

{

buttons = new Button[5];

setComponentOrientation(ComponentOrientation.RIGHT\_TO\_LEFT);  // comment

for (int i = 0; i < 5; i++)

{

  buttons[i] = new Button ("Button " + (i + 1));

  add (buttons[i]);

}

setLayout (new BoxLayout(this, BoxLayout.LINE\_AXIS));

setSize(400, 400);

setVisible(true);

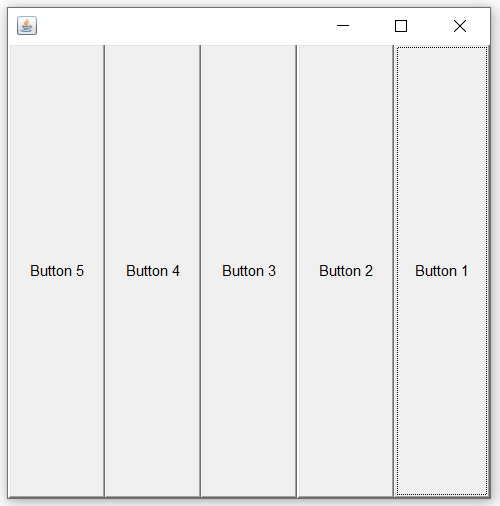
}

public static void main(String argvs[])

{

BoxLayoutExample3 obj = new BoxLayoutExample3();

}    }



* **GridBagLayout**: It is a powerful layout which arranges all the components in a grid of cells and maintains the aspect ration of the object whenever the container is resized. In this layout, cells may be different in size. It assigns a consistent horizontal and vertical gap among components. It allows us to specify a default alignment for components within the columns or rows.
* A **SpringLayout** arranges the children of its associated container according to a set of constraints. Constraints are nothing but horizontal and vertical distance between two-component edges. Every constraint is represented by a SpringLayout.Constraint object.

Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them.

Each edge position is dependent on the position of the other edge. If a constraint is added to create a new edge, than the previous binding is discarded. SpringLayout doesn't automatically set the location of the components it manages.

* **GroupLayout** groups its components and places them in a Container hierarchically. The grouping is done by instances of the Group class.

Group is an abstract class, and two concrete classes which implement this Group class are SequentialGroup and ParallelGroup.

SequentialGroup positions its child sequentially one after another whereas ParallelGroup aligns its child on top of each other.

The GroupLayout class provides methods such as createParallelGroup() and createSequentialGroup() to create groups.

#### JButton

JButton class is used to create a push-button on the UI. The button can contain some display text or image. It generates an event when clicked and double-clicked. A [JButton](https://www.educba.com/jbutton-in-java/) can be implemented in the application by calling one of its constructors.

**Example:**

JButton okBtn = new JButton(“Click Here”);

#### JLabel

[JLabel](https://www.educba.com/jlabel-in-java/) class is used to render a read-only text label or images on the UI. It does not generate any event.

**Example:**

JLabel textLbl = new JLabel(“This is a text label.”);

#### JTextField

[JTextField](https://www.educba.com/jtextfield-in-java/) renders an editable single-line text box. A user can input non-formatted text in the box. To initialize the text field, call its constructor and pass an optional integer parameter to it. This parameter sets the width of the box measured by the number of columns. It does not limit the number of characters that can be input in the box.

**Example:**

JTextField txtBox = new JTextField(20);

#### JTextArea

[JTextArea](https://www.educba.com/jtextarea-in-java/) class renders a multi-line text box. Similar to the JTextField, a user can input non-formatted text in the field. The constructor for JTextArea also expects two integer parameters which define the height and width of the text-area in columns. It does not restrict the number of characters that the user can input in the text-area.

**Example:**

JTextArea txtArea = new JTextArea(“This text is default text for text area.”, 5, 20);

The above code renders a multi-line text-area of height 5 rows and width 20 columns, with default text initialized in the text-area.

#### JPasswordField

[JPasswordField](https://www.educba.com/jpasswordfield/) is a subclass of JTextField class. It renders a text-box that masks the [user input](https://www.educba.com/java-user-input/) text with bullet points. This is used for inserting passwords into the application.

**Example:**

JPasswordField pwdField = new JPasswordField(15);  
var pwdValue = pwdField.getPassword();

It returns a password field of 15 column width. The getPassword method gets the value entered by the user.

#### JCheckBox

JCheckBox renders a check-box with a label. The check-box has two states – on/off. When selected, the state is on and a small tick is displayed in the box.

**Example:**

CheckBox chkBox = new JCheckBox(“Show Help”, true);

It returns a checkbox with the label Show Help. Notice the second parameter in the constructor. It is a boolean value that indicates the default state of the check-box. True means the check-box is defaulted to on state.

#### JRadioButton

JRadioButton is used to render a group of [radio](https://www.educba.com/javafx-radio-button/) buttons in the UI. A user can select one choice from the group.

**Example:**

ButtonGroup radioGroup = new ButtonGroup();  
JRadioButton rb1 = new JRadioButton(“Easy”, true);  
JRadioButton rb2 = new JRadioButton(“Medium”);  
JRadioButton rb3 = new JRadioButton(“Hard”);  
radioGroup.add(rb1);  
radioGroup.add(rb2);  
radioGroup.add(rb3);

The above code creates a button group and three radio button elements. All three elements are then added to the group. This ensures that only one option out of the available options in the group can be selected at a time. The default selected option is set to Easy.

#### JList

JList component renders a scrollable list of elements. A user can select a value or multiple values from the list. This select behavior is defined in the code by the developer.

**Example:**

#### String[] imtesD = { "Amreli", "Surat", "Baroda", "Mumbai" };

#### JList<String> list = new JList<>(imtesD);

#### list.setBounds(150,100, 75,75);

#### f.add(list);

#### JComboBox

[JComboBox](https://www.educba.com/jcombobox-in-java/) class is used to render a dropdown of the list of options.

**Example:**

String[] cityStrings = { "Mumbai", "London", "New York", "Sydney", "Tokyo" };  
JComboBox cities = new JComboBox(cityStrings);  
cities.setSelectedIndex(3);

The default selected option can be specified through the setSelectedIndex method. The above code sets Sydney as the default selected option.

## JMenu

JMenu, JMenuBar, and JMenuItems are a piece of Java Swing bundle. JMenuBar is an execution of a menu bar. The JMenuBar contains at least one JMenu objects. When these objects are chosen they show a popup demonstrating at least one JMenuItems. It essentially speaks to a menu. It contains a few JMenuItem Objects. It might likewise contain JMenu Objects (or submenu). The Menu class speaks to the drawdown menu part which is sent from a menu bar. The JMenuItem class speaks to the real thing in a menu. All things in a menu ought to get from class JMenuItem or one of its subclasses.

#### JFileChooser

[JFileChooser](https://www.educba.com/jfilechooser-in-java/) class renders a file selection utility. This component lets a user select a file from the local system.

**Example:**

package jmenudemo;

import javax.swing.\*;

import java.awt.event.\*;

import java.io.\*;

public class JMenuDemo extends JFrame implements ActionListener{

JMenuBar mb;

JMenu file;

JMenuItem open;

JTextArea ta;

JMenuDemo(){

open=new JMenuItem("Open File");

open.addActionListener(this);

file=new JMenu("File");

file.add(open);

mb=new JMenuBar();

mb.setBounds(0,0,800,20);

mb.add(file);

ta=new JTextArea(800,800);

ta.setBounds(0,20,800,800);

add(mb);

add(ta);

}

public void actionPerformed(ActionEvent e) {

if(e.getSource()==open){

JFileChooser fc=new JFileChooser();

int i=fc.showOpenDialog(this);

if(i==JFileChooser.APPROVE\_OPTION){

File f=fc.getSelectedFile();

String filepath=f.getPath();

try{

BufferedReader br=new BufferedReader(new FileReader(filepath));

String s1="",s2="";

while((s1=br.readLine())!=null){

s2+=s1+"\n";

}

ta.setText(s2);

br.close();

}catch (Exception ex) {ex.printStackTrace(); }

}

}

}

public static void main(String[] args) {

JMenuDemo om=new JMenuDemo();

om.setSize(500,500);

om.setLayout(null);

om.setVisible(true);

om.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

**Event Handling in Java**

An **event**can be defined as changing the state of an object or behavior by performing actions. Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.

The **java.awt.event** package can be used to provide various event classes.

### Classification of Events

* Foreground Events
* Background Events

#### 1. Foreground Events

Foreground events are the events that require user interaction to generate, i.e., foreground events are generated due to interaction by the user on components in Graphic User Interface (**GUI**). Interactions are nothing but clicking on a button, scrolling the scroll bar, cursor moments, etc.

#### 2. Background Events

Events that don’t require interactions of users to generate are known as background events. Examples of these events are operating system failures/interrupts, etc.

## Event Handling

It is a mechanism to **control the events** and to **decide what should happen after an event** occur. To handle the events, Java follows the ***Delegation Event model.***

### Delegation Event model

* It has Sources and Listeners.
* **Source:** Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, etc., to generate events.
* **Listeners:** Listeners are used for handling the events generated from the source. Each of these listeners represents interfaces that are responsible for handling events.

| **Event Class** | **Listener Interface** | **Description** |
| --- | --- | --- |
| ActionEvent | ActionListener | An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list. |
| AdjustmentEvent | AdjustmentListener | The adjustment event is emitted by an Adjustable object like Scrollbar. |
| FocusEvent | FocusListener | These are focus-related events, which include focus, focusin, focusout, and blur. |
| ItemEvent | ItemListener | An event that indicates whether an item was selected or not. checkbox |
| KeyEvent | KeyListener | An event that occurs due to a sequence of keypresses on the keyboard. |
| MouseEvent | MouseListener & MouseMotionListener | The events that occur due to the user interaction with the mouse (Pointing Device). |
| TextEvent | TextListener | An event that occurs when an object’s text changes. |
| WindowEvent | WindowListener | An event which indicates whether a window has changed its status or not. |

Different interfaces consists of different methods which are specified below.

| **Listener Interface** | **Methods** |
| --- | --- |
| ActionListener | * actionPerformed() |
| AdjustmentListener | * adjustmentValueChanged() |
| FocusListener | * focusGained() * focusLost() |
| ItemListener | * itemStateChanged() |
| KeyListener | * keyTyped() * keyPressed() * keyReleased() |
| MouseListener | * mousePressed() * mouseClicked() * mouseEntered() * mouseExited() * mouseReleased() |
| MouseMotionListener | * mouseMoved() * mouseDragged() |
| TextListener | * textChanged() |
| WindowListener | * windowActivated() * windowDeactivated() * windowOpened() * windowClosed() * windowClosing() |

# Java ActionListener Interface

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event [package](https://www.javatpoint.com/package). It has only one method: actionPerformed().

## actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

## How to write ActionListener

The common approach is to implement the ActionListener. If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

1. public class ActionListenerExample Implements ActionListener

2) Register the component with the Listener:

1. component.addActionListener(instanceOfListenerclass);

3) Override the actionPerformed() method:

1. public void actionPerformed(ActionEvent e){
2. //Write the code here
3. }

## Java ActionListener Example: On Button click

import java.awt.\*;

import java.awt.event.\*;

public class ActionListenerExample {

public static void main(String[] args) {

    Frame f=new Frame("ActionListener 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 Event.");

    }

    });

    f.add(b);f.add(tf);

    f.setSize(400,400);

    f.setLayout(null);

    f.setVisible(true);

}

}

# ItemListener Interface

The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

## itemStateChanged() method

The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

import java.awt.\*;

import java.awt.event.\*;

public class ItemListenerExample implements ItemListener{

    Checkbox checkBox1,checkBox2;

    Label label;

    ItemListenerExample(){

        Frame f= new Frame("CheckBox Example");

        label = new Label();

        label.setAlignment(Label.CENTER);

        label.setSize(400,100);

        checkBox1 = new Checkbox("C++");

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

        checkBox2 = new Checkbox("Java");

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

        f.add(checkBox1); f.add(checkBox2); f.add(label);

        checkBox1.addItemListener(this);

        checkBox2.addItemListener(this);

        f.setSize(400,400);

        f.setLayout(null);

        f.setVisible(true);

     }

    public void itemStateChanged(ItemEvent e) {

        if(e.getSource()==checkBox1)

            label.setText("C++ Checkbox: "

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

        if(e.getSource()==checkBox2)

        label.setText("Java Checkbox: "

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

     }

public static void main(String args[])

{

    new ItemListenerExample();

}

}

**TextListner :** The class which processes the TextEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addTextListener() method.

## Interface declaration

Following is the declaration for **java.awt.event.TextListener** interface:

public interface TextListener

extends EventListener

## Interface methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **void textValueChanged(TextEvent e)**  Invoked when the value of the text has changed. |

## Methods inherited

This interface inherits methods from the following interfaces:

* java.awt.EventListener

## TextListener Example

Create the following java program using any editor of your choice in say **D:/ > AWT > com > tutorialspoint > gui >**

*AwtListenerDemo.java*

package textlistnerdemo;

import java.awt.\*;

import java.awt.event.\*;

public class TextListnerDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

private TextField textField;

public TextListnerDemo(){

prepareGUI();

}

public static void main(String[] args){

TextListnerDemo awtListenerDemo = new TextListnerDemo();

awtListenerDemo.showTextListenerDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showTextListenerDemo(){

headerLabel.setText("Listener in action: TextListener");

textField = new TextField(10);

textField.addTextListener(new CustomTextListener());

Button okButton = new Button("OK");

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

statusLabel.setText("Entered text=: "

+ textField.getText());

}

});

controlPanel.add(textField);

controlPanel.add(okButton);

mainFrame.setVisible(true);

}

class CustomTextListener implements TextListener {

public void textValueChanged(TextEvent e) {

statusLabel.setText("Entered text: " + textField.getText());

}

}

}

**FocusListener:** it is used for receiving keyboard focus events. The class that process focus events needs to implements this interface.

## Class declaration

Following is the declaration for **java.awt.event.FocusListener** interface:

public interface FocusListener

extends EventListener

|  |  |
| --- | --- |
| 1 | **void focusGained(FocusEvent e)**  Invoked when a component gains the keyboard focus. |
| 2 | **void focusLost(FocusEvent e)**  Invoked when a component loses the keyboard focus. |

*AwtListenerDemo.java*

import java.awt.\*;

import java.awt.event.\*;

public class FocusListenerDemo {

private Frame mainFrame;

private Label headerLabel;

private Label statusLabel;

private Panel controlPanel;

public FocusListenerDemo(){

prepareGUI();

}

public static void main(String[] args){

FocusListenerDemo awtListenerDemo = new FocusListenerDemo ();

awtListenerDemo.showFocusListenerDemo();

}

private void prepareGUI(){

mainFrame = new Frame("Java AWT Examples");

mainFrame.setSize(400,400);

mainFrame.setLayout(new GridLayout(3, 1));

mainFrame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent windowEvent){

System.exit(0);

}

});

headerLabel = new Label();

headerLabel.setAlignment(Label.CENTER);

statusLabel = new Label();

statusLabel.setAlignment(Label.CENTER);

statusLabel.setSize(350,100);

controlPanel = new Panel();

controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);

mainFrame.add(controlPanel);

mainFrame.add(statusLabel);

mainFrame.setVisible(true);

}

private void showFocusListenerDemo(){

headerLabel.setText("Listener in action: FocusListener");

Button okButton = new Button("OK");

Button cancelButton = new Button("Cancel");

okButton.addFocusListener(new CustomFocusListener());

cancelButton.addFocusListener(new CustomFocusListener());

controlPanel.add(okButton);

controlPanel.add(cancelButton);

mainFrame.setVisible(true);

}

class CustomFocusListener implements FocusListener{

public void focusGained(FocusEvent e) {

statusLabel.setText(statusLabel.getText()

+ e.getComponent().getClass().getSimpleName() + " gained focus. ");

}

public void focusLost(FocusEvent e) {

statusLabel.setText(statusLabel.getText()

+ e.getComponent().getClass().getSimpleName() + " lost focus. ");

}

}

}

**AdjustmentListener**: is used for receiving adjustment events. The class that process adjustment events needs to implements this interface.

## Class declaration

Following is the declaration for **java.awt.event.AdjustmentListener** interface:

public interface AdjustmentListener

extends EventListener

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class AdjustmentListenerExample implements AdjustmentListener

{

Frame jf;

Panel jp, jp2;

Label frameLabel1;

AdjustmentListenerExample()

{

jf = new JFrame("Scrollbar");

//Creating the first JPanel and adding two JLabels to it

jp = new Panel();

//Creating a Label

Label panelLabel1 = new Label("Handling a Scrollbar drag event", Label.CENTER);

jp = new Panel(new BorderLayout());

//Adding the Label to NORTH of the Panel

jp.add(panelLabel1,BorderLayout.NORTH);

//Creating the horizontal Scrollbar

Scrollbar scrollBHorizontal = new Scrollbar(Scrollbar.HORIZONTAL, 10, 40, 0, 100);

//Creating the vertical Scrollbar

Scrollbar scrollBVertical = new Scrollbar(Scrollbar.VERTICAL, 10, 60, 0, 100);

//Adding the horizontal Scrollbar to SOUTH of Panel

jp.add(scrollBHorizontal,BorderLayout.SOUTH);

//Adding the vetical Scrollbar to EAST of JPanel

jp.add(scrollBVertical, BorderLayout.EAST);

//Getting the current position value of horizontal scrollbar

Integer i = scrollBHorizontal.getValue();

//Creating a JLabel and setting its value to the current position value of horizontal scrollbar.

frameLabel1 = new Label(i.toString());

//Adding this JLabel to SOUTH of the JFrame

jf.add(frameLabel1, BorderLayout.SOUTH);

//Adding the first JPanel to the CENTER of JFrame

jf.add(jp,BorderLayout.CENTER);

//Registering class ScrollEx2 to catch and respond to scrollbar adjustment events

scrollBHorizontal.addAdjustmentListener(this);

scrollBVertical.addAdjustmentListener(this);

jf.setSize(350,270);

jf.setVisible(true);

}

public void adjustmentValueChanged(AdjustmentEvent e)

{

Integer i =e.getValue();

frameLabel1.setText(i.toString());

}

public static void main(String[] ar)

{

new AdjustmentListenerExample();

}

}