JFrame frame = new JFrame();

Container container = frame.getContentPane();

// Create the layout

GridBagLayout gbl = new GridBagLayout();

// Set layout on container

container.setLayout(gbl);

// Place a component at cell location (1,1)

GridBagConstraints gbc = new GridBagConstraints();

gbc.gridx = 1;

gbc.gridy = 1;

// Add other gridbag constraints here

// Associate the gridbag constraints with the component

gbl.setConstraints(*component*, gbc);

// Add the component to the container

container.add(*component*);

// Show the frame

frame.pack();

frame.setVisible(true);

Typical code using this manager is

GridBagLayout gridbag = new GridBagLayout();

setLayout(gridbag);

GridBagConstraints constraints = new GridBagConstraints();

// set values in constraints ...

Button btn = new Button("Hello");

add(btn);

// tell the layout manager of the constraints

gridbag.setConstraints(btn, constraints);

The fields of GridBagConstraints are

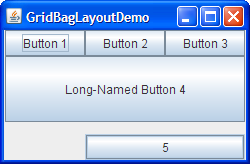
public int gridx, gridy, gridwidth, gridheight;

public double weightx, weighty;

public int anchor, fill;

public Insets insets;

public int ipadx, ipady



The following code creates the GridBagLayout and the components it manages. You can find the entire source file in [GridBagLayoutDemo.java](http://da2i.univ-lille1.fr/doc/tutorial-java/uiswing/layout/examples/GridBagLayoutDemo.java).

JButton button;

pane.setLayout(new GridBagLayout());

GridBagConstraints c = new GridBagConstraints();

c.fill = GridBagConstraints.HORIZONTAL;

button = new JButton("Button 1");

c.weightx = 0.5;

c.gridx = 0;

c.gridy = 0;

pane.add(button, c);

button = new JButton("Button 2");

c.gridx = 1;

c.gridy = 0;

pane.add(button, c);

button = new JButton("Button 3");

c.gridx = 2;

c.gridy = 0;

pane.add(button, c);

button = new JButton("Long-Named Button 4");

c.ipady = 40; //make this component tall

c.weightx = 0.0;

c.gridwidth = 3;

c.gridx = 0;

c.gridy = 1;

pane.add(button, c);

button = new JButton("5");

c.ipady = 0; //reset to default

c.weighty = 1.0; //request any extra vertical space

c.anchor = GridBagConstraints.PAGE\_END; //bottom of space

c.insets = new Insets(10,0,0,0); //top padding

c.gridx = 1; //aligned with button 2

c.gridwidth = 2; //2 columns wide

c.gridy = 2; //third row

pane.add(button, c);

Notes:-

### GridBagConstraints

The following is a complete list of all of the constraints:

* anchor determines position in the display area
* fill determines if a component is stretched to fill the area
* gridheight and gridwidth determine the number of rows and columns in the component's area to span
* gridx and gridy determine the position of the component's area.
* insets determine a border around a component's area.
* ipadx and ipady allows the minimum or preferred size of a component to be adjusted.
* weightx and weighty determine **the sizes of the rows and columns** of the grids.

The anchor attribute indicates where in the grid the component will appear in its area if it does not exactly fill its area.

The fill attribute determines whether the components should be stretched to fill the entire area.

The weight attributes determine the various sizes of the grid areas.

**1)gridx, gridy**

Use these variables to specify explicitly where on the grid the layout manager should place the component.

The upper-left cell is the origin, which has the location gridx = 0, gridy = 0. The default value is GridBagConstraints.RELATIVE, The default value RELATIVE means that this component is placed next to / below the previously added component (this is usually not recommended)

We recommend specifying the gridx and gridy values for each component; this tends to result in more predictable layouts.

**gridwidth, gridheight**

Specify the number of columns (for gridwidth) or rows (for gridheight) that component should span.Use gridwidth and gridheight to specify the size of the components display area. You specify this size in number of grid cells.

For example, the values gridwidth = 2 and gridheight = 1 mean that the component display area is two cells wide and one cell high in the grid. The default value of gridwidth and gridheight is 1.

To specify that a component is the last one in its row or column, you can set gridwidth and gridheight to GridBagConstraints.REMAINDER. You can use the value GridBagConstraints.RELATIVE to indicate that the component is next to the last one in the row or column.

**Note:** GridBagLayout doesn't allow components to span multiple rows unless the component is in the leftmost column or you've specified positive gridx and gridy values for the component

**fill**

Use fill to specify how GridBagLayout should resize components when the display area is larger than the component. Set fill to GridBagConstraints.HORIZONTAL to make the component sufficiently wide to fill its display area (without changing the component height). Set fill to GridBagConstraints.VERTICAL to make the component sufficiently tall to fill its display area (without changing the component width). Set fill to GridBagConstraints.BOTH to make the component fill the display area completely. Thus, the value GridBagConstraints.BOTH is a combination of GridBagConstraints.HORIZONTAL and GridBagConstraints.VERTICAL. The default value of fill is GridBagConstraints.NONE, which specifies no fill for the component.

**anchor**

Use anchor to specify where a component should be placed if it is smaller than the display area. GridBagLayout attaches the component to the specified location. The following are the possible values:

   GridBagConstraints.CENTER (default)  
   GridBagConstraints.NORTH  
   GridBagConstraints.NORTHEAST  
   GridBagConstraints.EAST  
   GridBagConstraints.SOUTHEAST  
   GridBagConstraints.SOUTH  
   GridBagConstraints.SOUTHWEST  
   GridBagConstraints.WEST  
   GridBagConstraints.NORTHWEST

**ipadx, ipady**

Use ipadx and ipady to enlarge the minimum size of components. GridBagLayout adds ipadx pixels to the left and right of the minimum size of the component. Similarly, GridBagLayout adds ipady pixels to the bottom and top of the minimum size of the component. Thus, GridBagLayout increases the minimum width and height by ipadx\*2 and ipady\*2 pixels, respectively. This **ipadx**, **ipady** Specifies the internal padding: how much to add to the minimum size of the component. The default value is zero.

**Insets**

Use Insets to specify the minimum border between the component and its display area. The value must be an instance of the class Insets. You can use the constructor Insets(int, int, int, int) to create an Insets instance with top, left, bottom, and right insets. GridBagLayout then inserts the specified space between the edges of the component and its display area.

**insets**

Specifies the external padding of the component -- the minimum amount of space between the component and the edges of its display area. The value is specified as an [Insets](http://java.sun.com/j2se/5.0/docs/api/java/awt/Insets.html)[ (in the API reference documentation)](http://java.sun.com/j2se/5.0/docs/api/java/awt/Insets.html) object. By default, each component has no external padding

-------------------------------------------------

|FIRST\_LINE\_START PAGE\_START FIRST\_LINE\_END|

| |

| |

|LINE\_START CENTER LINE\_END|

| |

| |

|LAST\_LINE\_START PAGE\_END LAST\_LINE\_END|

-------------------------------------------------

**weightx**, **weighty**

Unless you specify at least one nonzero value for weightx or weighty, all the components clump together in the center of their container. This is because when the weight is 0.0 (the default), the GridBagLayout puts any extra space between its grid of cells and the edges of the container.

Generally weights are specified with 0.0 and 1.0 as the extremes: the numbers in between are used as necessary. Larger numbers indicate that the component's row or column should get more space.

**weightx, weighty**

Use weightx and weighty to specify how GridBagLayout should distribute space. You can use numeric values for weightx and weighty to distribute space among columns (weightx) and rows (weighty). These weights determine how much extra space a row (or column) will get when the container expands. By setting the weightx and weighty values, you control how rows and columns scale. Rows (columns) with larger weights will grow faster than rows (columns) with smaller weights. Typically, weightx and weighty have values between 0.0 and 1.0. The default weight is zero (0.0), which means no growth. When all weights are zero, GridBagLayout places the components together at the center of the container. Thus, GridBagLayout puts space between the grid and the edges of the container. Note that the actual weight for each row (column) is a combination of the weights of each of the components in the row (column).

You may find the task of setting up these variables difficult. If you start modifying the values without a clear idea of how they affect the layout, you may find it difficult to get the layout and resizing behavior you want. The key to successful layout creation is to plan ahead and to design the layout before specifying it.

fill Fill specifies how the component should expand within its display area if the area width/height is larger than its preferred size.

GridBagConstraints.NONE // Can't expand (Default)

GridBagConstraints.VERTICAL // Expand vertically

GridBagConstraints.HORIZONTAL // Expand horizontally

GridBagConstraints.BOTH // Expand vertically and horizontally

insets A java.awt.Insets object adds padding space to the component. Insets should be rarely used because they often produce bad results, with the exception of JPanels and JLabels. The constructor parameters (in pixels) specify the top, left, bottom, right. For example,

gbc.insets = new Insets(10, 5, 10, 4);

Default value: Insets(0,0,0,0).