Layout Managers

Ji Zhenyan & Åke Malmberg

Using Layout Managers

- Every container, by default, has a layout manager -- an object that implements the LayoutManager interface.
- If a container's default layout manager doesn't suit your needs, you can easily replace it with another one.
- As a rule, the only time you have to think about layout managers is when you create a JPanel or add components to a content pane .
- When you add components to a panel or content pane, the arguments you specify to the add method depend on the layout manager that the panel or content pane is using.

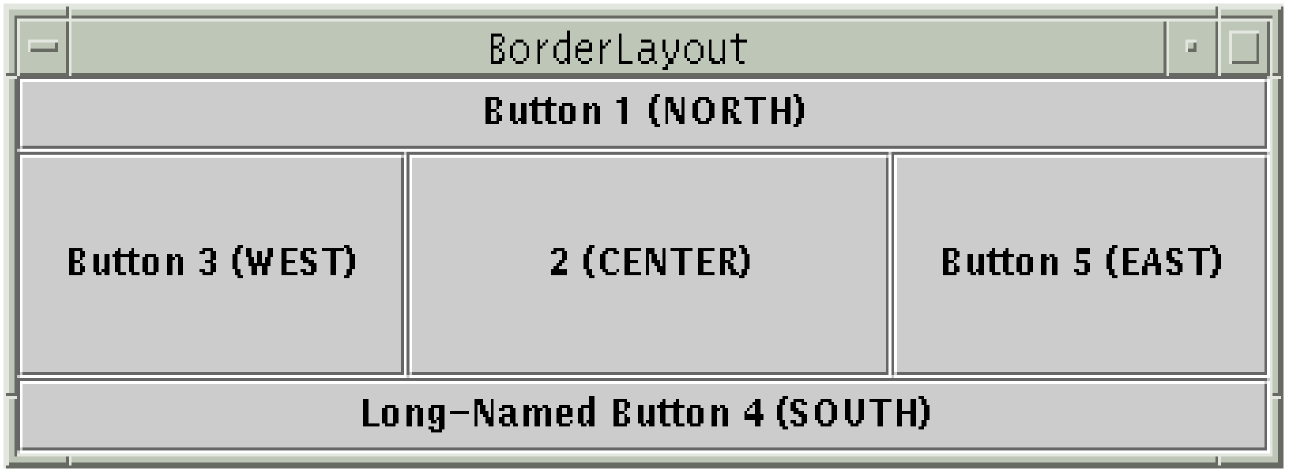
- display a component in as much space as it can get. using BorderLayout or GridBagLayout. If use BorderLayout, need to put the space-hungry component in the center. With GridBagLayout, need to set the constraints for the component so that fill=GridBagConstraints.BOTH.
- display a few components in a compact row at their natural size. using either FlowLayout manager or the BoxLayout manager.
- display a few components of the same size in rows and columns. GridLayout
- display a few components in a row or column, possibly with varying amounts of space between them, custom alignment, or custom component sizes. BoxLayout
- a complex layout with many components. using GridBagLayout or grouping the components into one or more JPanels to simplify layout.

- If use a container's default layout manager, you don't have to do a thing. The constructor for the container creates a layout manager instance and initializes the container to use it.
- To use a layout manager other than the default layout manager, you must create an instance of the desired layout manager class and tell the container to use it.
- The **content pane** provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame.
- Container contentPane = getContentPane(); contentPane.setLayout(new FlowLayout());

How to Use BorderLayout

- BorderLayout is the default layout manager for every content pane. (As described in Using Top-Level Containers , the content pane is the main container in all frames, applets, and dialogs.)
- A BorderLayout has five areas available to hold components: north, south, east, west, and center. All extra space is placed in the center area.

BorderWindow.java

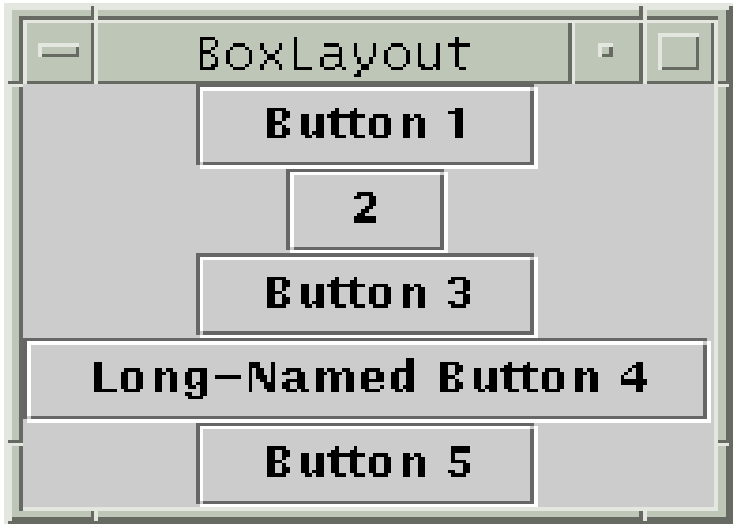


- By default, a BorderLayout puts no gap between the components it manages.
- In the preceding applet, any apparent gaps are the result of the buttons reserving extra space around their apparent display area.
- specify gaps (in pixels) using the following constructor:
- BorderLayout(int *horizontalGap*, int *verticalGap*)
- Can also use the following methods to set the horizontal and vertical gaps, respectively:
- void setHgap(int)
- void setVgap(int)
- BorderWindow.java

How to Use BoxLayout

- The BoxLayout class puts components in a single row or column. It respects the components' requested maximum sizes, and also lets you align components.
- BoxLayout either stacks its components on top of each other (with the first component at the top) or places them in a tight row from left to right -- your choice. You might think of it as a full-featured version of FlowLayout.

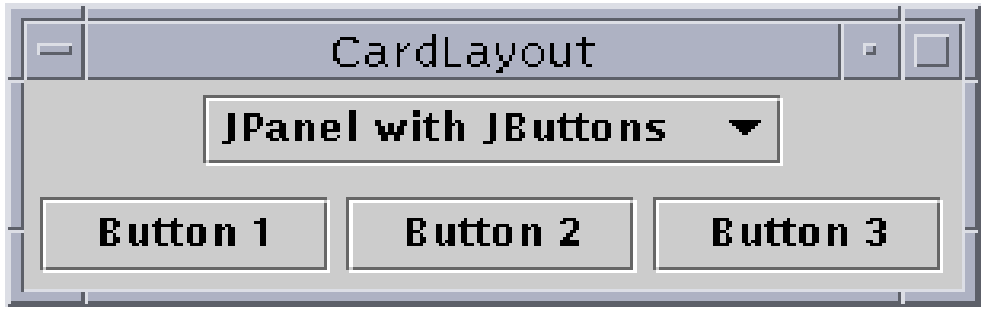
BoxWindow.java



How to Use CardLayout

- The CardLayout class lets you implement an area that contains different components at different times. Tabbed panes are intermediate Swing containers that provide similar functionality, but with a pre-defined GUI. A CardLayout is often controlled by a combo box , with the state of the combo box determining which panel (group of components) the CardLayout displays.
- the CardLayout class helps you manage two or more components (usually JPanel instances) that share the same display space. Another way to accomplish the same thing is to use a tabbed pane

CardWindow.java

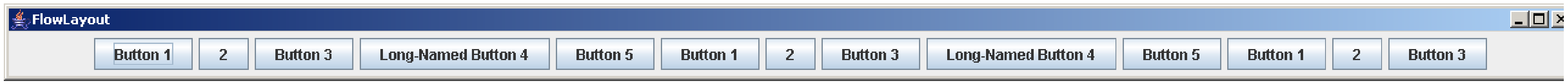


How to Use FlowLayout

- FlowLayout is the default layout manager for every JPanel. It simply lays out components from left to right, starting new rows if necessary.
- FlowLayout puts components in a row, sized at their preferred size.
- If the horizontal space in the container is too small to put all the components in one row, FlowLayout uses multiple rows.
- Within each row, components are centered (the default), left-aligned, or right-aligned as specified when the FlowLayout is created.

- The FlowLayout class has three constructors:
- public FlowLayout()
- public FlowLayout(int *alignment*)
- public FlowLayout(int *alignment*, int *horizontalGap*, int *verticalGap*)
- *Alignment*----FlowLayout.LEFT, FlowLayout.CENTER, or FlowLayout.RIGHT.
- The *horizontalGap* and *verticalGap* arguments specify the number of pixels to put between components. If you don't specify a gap value, FlowLayout uses 5 for the default gap value.

FlowWindow.java

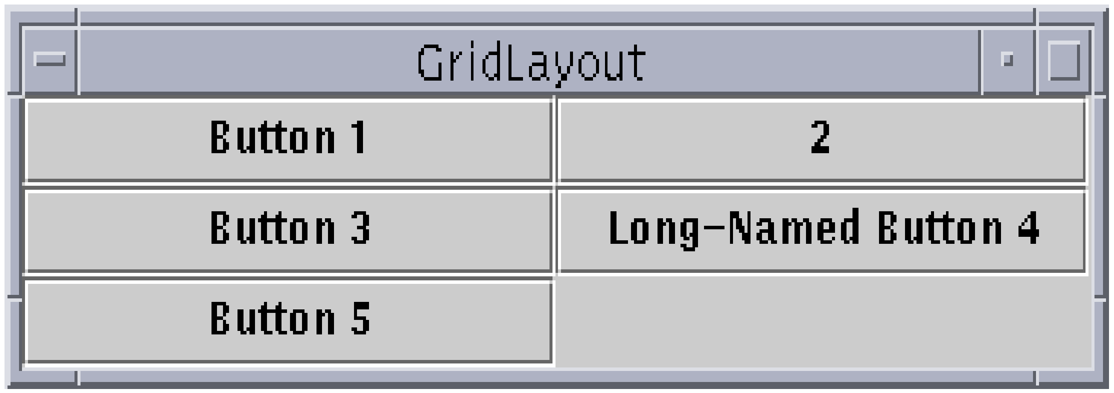


How to Use GridLayout

- GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.
- A GridLayout places components in a grid of cells. Each component takes all the available space within its cell, and each cell is exactly the same size. If you resize the GridLayout window, you'll see that the GridLayout changes the cell size so that the cells are as large as possible, given the space available to the container.

- The GridLayout class has two constructors:
- public GridLayout(int *rows*, int *columns*)
- public GridLayout(int *rows*, int *columns*, int *horizontalGap*, int *verticalGap*)
- At least one of the *rows* and *columns* arguments must be nonzero.
- The *horizontalGap* and *verticalGap* arguments specify the number of pixels between cells. If you don't specify gaps, their values default to zero.
- CellWindow.java

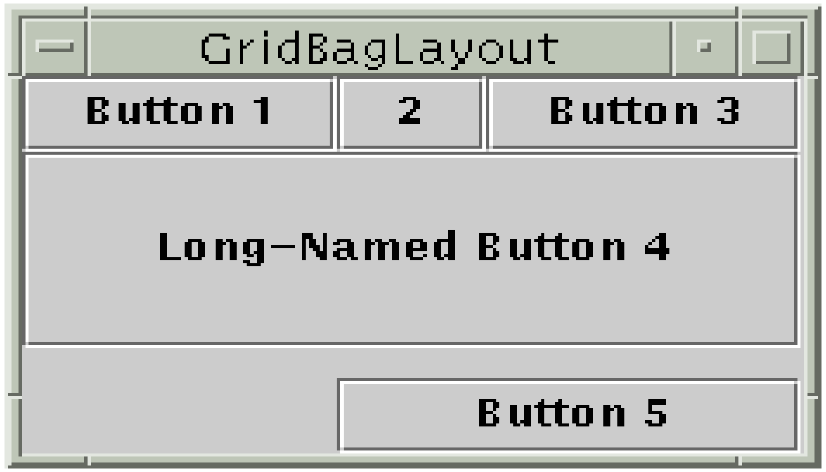
GridWindow.java



How to Use GridBagLayout

- the most flexible -- and complex -- layout manager.
- A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns. Not all rows necessarily have the same height. Similarly, not all columns necessarily have the same width.

GridBagWindow.java



- GridBagConstraints instance variables:
- **gridx, gridy**
  - Specify the row and column at the upper left of the component.
  - The leftmost column has address gridx=0 and the top row has address gridy=0.
  - Use GridBagConstraints.RELATIVE (the default value) to specify that the component be placed just to the right of (for gridx) or just below (for gridy) the component that was added to the container just before this component was added.
  - 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) in the component's display area. These constraints specify the number of cells the component uses, *not* the number of pixels it uses. The default value is 1.
  - GridBagConstraints.REMAINDER specify that the component be the last one in its row (for gridwidth) or column (for gridheight).
  - GridBagConstraints.RELATIVE specify that the component be the next to last one in its row (for gridwidth) or column (for gridheight).
  - **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**
  - Used when the component's display area is larger than the component's requested size to determine whether and how to resize the component.
  - Valid values (defined as GridBagConstraints constants) are NONE (the default), HORIZONTAL (make the component wide enough to fill its display area horizontally, but don't change its height), VERTICAL (make the component tall enough to fill its display area vertically, but don't change its width), and BOTH (make the component fill its display area entirely).

- **ipadx, ipady**
  - Specifies the internal padding: how much to add to the minimum size of the component. The default value is zero.
  - The width of the component will be at least its minimum width plus ipadx*2 pixels, since the padding applies to both sides of the component.
  - the height of the component will be at least its minimum height plus ipady*2 pixels.
- **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 object. By default, each component has no external padding.

- **anchor**
  - Used when the component is smaller than its display area to determine where (within the area) to place the component.
  - Valid values (defined as GridBagConstraints constants) are CENTER (the default), NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, and NORTHWEST.
- **weightx**, **weighty**
  - Weights are used to determine how to distribute space among columns (weightx) and among rows (weighty); this is important for specifying resizing behavior.

- 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. For each column, the weight is related to the highest weightx specified for a component within that column, with each multicolumn component's weight being split somehow between the columns the component is in. Similarly, each row's weight is related to the highest weighty specified for a component within that row. Extra space tends to go toward the rightmost column and bottom row.