Supplement IV.F: GridBagLayout

For Introduction to Java Programming
Y. Daniel Liang

The GridBagLayout manager is the most flexible and the most complex. It is similar to the GridLayout manager in the sense that both layout managers arrange components in a grid. The components of GridBagLayout can vary in size, however, and can be added in any order. For example, with GridBagLayout you can create the layout shown in Figure 31.5.

******Same as Fig28.5 in intro6e p924***

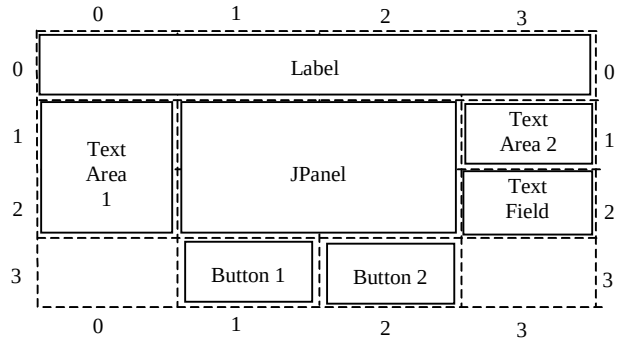

Figure 31.5

A GridBagLayout manager divides the container into cells. A component can occupy several cells.

The constructor GridBagLayout() is used to create a new GridBagLayout. In GridLayout, the grid size (the number of rows and columns) may be specified in the constructor. It is not specified in GridBagLayout. The actual size is dynamically determined by the constraints associated with the components added to the container of GridBagLayout.

Each GridBagLayout uses a dynamic rectangular grid of cells, with each component occupying one or more cells called its *display area*. Each component managed by a GridBagLayout is associated with a GridBagConstraints instance that specifies how the component is laid out within its display area. How a GridBagLayout places a set of components depends on the GridBagConstraints and minimum size of each component, as well as the preferred size of the component's container.

To use GridBagLayout effectively, you must customize the GridBagConstraints of one or more of its components. You customize a GridBagConstraints object by setting one or more of its public instance variables. These variables specify the component's location, size, growth factor, anchor, inset, filling, and padding.

31.3.2.1 Location

The variables **gridx** and **gridy** specify the cell at the upper left of the component's display area, where the upper-leftmost cell has the address gridx=0, gridy=0. Note that gridx specifies the column in which the component will be placed, and gridy specifies the row in which it will be placed. In Figure 31.5, Button 1 has a gridx value of 1 and a gridy value of 3, and Label has a gridx value of 0 and a gridy value of 0.

You can assign GridBagConstraints.RELATIVE to gridx to specify that the component be placed immediately after the component that was just added to the container. You can assign GridBagConstraints.RELATIVE to gridy to specify that the component be placed immediately below the component that was just added to the container.

31.3.2.2 Size

The variables **gridwidth** and **gridheight** specify the number of cells in a row (for gridheight) or column (for gridwidth) in the component's display area. The default value is 1. In Figure 31.5, the JPanel in the center occupies two columns and two rows, so its gridwidth is 2, and its gridheight is 2. Text Area 2 occupies one row and one column; therefore its gridwidth is 1, and its gridheight is 1.

You can assign GridBagConstraints.RELATIVE.REMAINDER to gridwidth (gridheight) to specify that the component is to be the last one in its row (column).

31.3.2.3 Growth Weight

The variables **weightx** and **weighty** specify the extra horizontal and vertical space to allocate for the component when the resulting layout is smaller horizontally than the area it needs to fill.

The GridBagLayout manager calculates the weight of a column to be the maximum weightx (weighty) of all the components in a column (row). The extra space is distributed to each column (row) in proportion to its weight.

Unless you specify a weight for at least one component in a row (weightx) and a column (weighty), all the components clump together in the center of their container. This is because, when the weight is zero (the default), the GridBagLayout puts any extra space between its grid of cells and the edges of the container. You will see the effect of these parameters in Listing 31.2.

31.3.2.4 Anchor

The variable **anchor** specifies where in the area the component is placed when it does not fill the entire area. Valid values are:

GridBagConstraints.CENTER (the default)

GridBagConstraints.NORTH

GridBagConstraints.NORTHEAST

GridBagConstraints.EAST

GridBagConstraints.SOUTHEAST

GridBagConstraints.SOUTH

GridBagConstraints.SOUTHWEST

GridBagConstraints.WEST

GridBagConstraints.NORTHWEST

31.3.2.5 Filling

The variable **fill** specifies how the component should be resized if its viewing area is larger than its current size. Valid values are GridBagConstraints.NONE (the default), GridBagConstraints.HORIZONTAL (makes the component wide enough to fill its display area horizontally, but doesn't change its height), GridBagConstraints.VERTICAL (makes the component tall enough to fill its display area vertically, but doesn't change its width), and GridBagConstraints.BOTH (makes the component totally fill its display area).

31.3.2.6 Insets

The variable **insets** specifies the external padding of the component, the minimum amount of space between the component and the edges of its display area. The default value is new Insets(0, 0, 0, 0).

31.3.2.7 Padding

The variables **ipadx** and **ipady** specify the internal padding of the component: how much space to add to its minimum width and height. The width of the component is at least its minimum width plus (ipadx * 2) pixels, and the height of the component is at least its minimum height plus (ipady * 2) pixels. The default value of these variables is 0. The insets variable specifies the external padding, while the ipadx and ipady variables specify the internal padding, as shown in Figure 31.6.

******Same as Fig28.6 in intro6e p926***

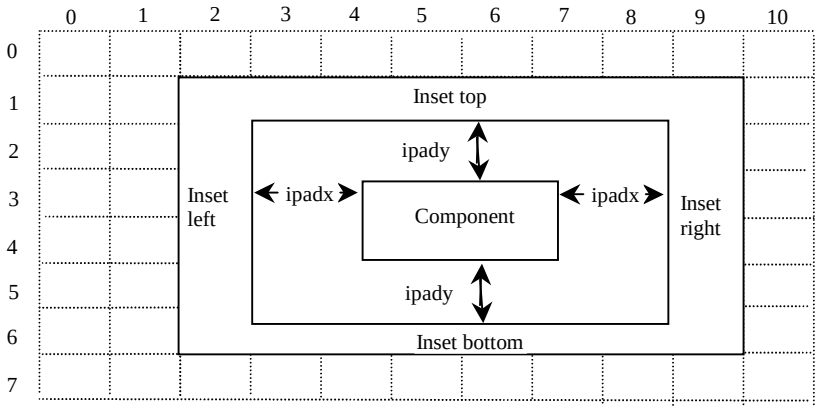

Figure 31.6

You can specify the external insets and internal padding for a component in the container of the GridBadLayout manager.

31.3.2.8 Constructing a GridBagConstraints Object

There are two constructors for creating a GridBagConstraints object:

[BL] **public** GridBagConstraints()

Constructs a GridBagConstraints object with all of its fields set to their default values.

[BL] **public** GridBagConstraints(**int** gridx, **int** gridy, **int** gridwidth, **int** gridheight, **double** weightx, **double** weighty, **int** anchor, **int** fill, Insets insets, **int** ipadx, **int** ipady)

Constructs a GridBagConstraints object with the specified field values.

31.3.2.9 Adding a Component to the Container of GridBagLayout

To add a component to the container of GridBarLayout, use the following method in the container:

public void add(Component comp, Object gbConstraints)

Adds a component to the container with the specified GridBagConstraints.

31.3.2.10 Example: Using GridBagLayout

Listing 31.2 gives a program that uses the GridBagLayout manager to create a layout for Figure 31.5. The output of the program is shown in Figure 31.7.

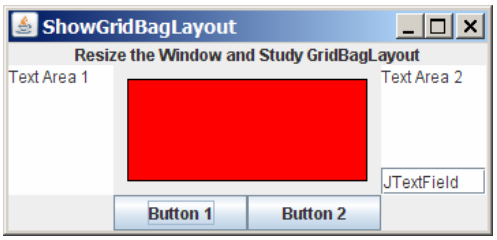

Figure 31.7

The components are placed in the frame of GridBagLayout.

Listing 31.2 ShowGridBagLayout.java

PD: Please add line numbers in the following code
***Layout: Please layout exactly. Don't skip the space. This is true for all source code in the book. Thanks, AU.
<Side Remark line 5: UI components>
<Side Remark line 15: create UI>
<Side Remark line 51: set constraints>
<Side Remark line 68: main omitted>***

```
import java.awt.*;
import javax.swing.*;

public class ShowGridBagLayout extends JApplet {
  private JLabel jlbl = new JLabel(
    "Resize the Window and Study GridBagLayout", JLabel.CENTER);
  private JTextArea jta1 = new JTextArea("Text Area 1", 5, 15 );
  private JTextArea jta2 = new JTextArea("Text Area 2", 5, 15 );
  private JTextField jtf = new JTextField("JTextField");
  private JPanel jp = new JPanel();
  private JButton jbt1 = new JButton("Button 1");
  private JButton jbt2 = new JButton("Button 2");

  public ShowGridBagLayout() {
    // Set GridBagLayout in the container
    setLayout(new GridBagLayout());

    // Create an GridBagConstraints object
    GridBagConstraints gbConstraints = new GridBagConstraints();

    gbConstraints.fill = GridBagConstraints.BOTH;
    gbConstraints.anchor = GridBagConstraints.CENTER;

    Container container = getContentPane();

    // Place JLabel to occupy row 0 (the first row)
    addComp(jlbl, container, gbConstraints, 0, 0, 1, 4, 0, 0);

    // Place text area 1 in row 1 and 2, and column 0
    addComp(jta1, container, gbConstraints, 1, 0, 2, 1, 5, 1);

    // Place text area 2 in row 1 and column 3
    addComp(jta2, container, gbConstraints, 1, 3, 1, 1, 5, 1);

    // Place text field in row 2 and column 3
    addComp(jtf, container, gbConstraints, 2, 3, 1, 1, 5, 0);
```

```
    // Place JButton 1 in row 3 and column 1
    addComp(jbt1, container, gbConstraints, 3, 1, 1, 1, 5, 0);

    // Place JButton 2 in row 3 and column 2
    addComp(jbt2, container, gbConstraints, 3, 2, 1, 1, 5, 0);

    // Place Panel in row 1 and 2, and column 1 and 2
    jp.setBackground(Color.red);
    jp.setBorder(new javax.swing.border.LineBorder(Color.black));
    gbConstraints.insets = new Insets(10, 10, 10, 10);
    addComp(jp, container, gbConstraints, 1, 1, 2, 2, 10, 1);
  }

  /** Add a component to the container of GridBagLayout */
  private void addComp(Component c, Container container,
                       GridBagConstraints gbConstraints,
                       int row, int column,
                       int numberOfRows,  int numberOfColumns,
                       double weightx, double weighty) {
    // Set parameters
    gbConstraints.gridx = column;
    gbConstraints.gridy = row;
    gbConstraints.gridwidth = numberOfColumns;
    gbConstraints.gridheight = numberOfRows;
    gbConstraints.weightx = weightx;
    gbConstraints.weighty = weighty;

    // Add component to the container with the specified layout
    container.add(c, gbConstraints);
  }
        }
```

The program defines the addComp method (lines 52–67) to add a component to the container of GridBagLayout with the specified constraints. The GridBagConstraints object gbConstraints created in line 19 is used to specify the layout constraints for each component. Before adding a component to the container, set the constraints in gbConstraints and then use container.add(c, gbConstraints) (line 66) to add the component to the container.

What would happen if you change the weightx parameter for jbt2 to 10 in line 42? Now jbt2's weightx is larger than jbt1's. When you enlarge the window, jbt2 will get larger horizontally than jbt1.

The weightx and weighty for all the other components are 0. Whether the size of these components grows or shrinks depends on the fill parameter. The program defines fill = BOTH for all the components added to the container (line 21).

Consider this scenario: Suppose that you enlarge the window. The display area for text area jta1 will increase. Because fill is BOTH for jta1, jta1 fills in its new display area. If you set fill to NONE for jta1, jta1 will not expand or

shrink when you resize the window.

The insets parameter is (0, 0, 0, 0) by default. For the panel jp, insets is set to (10, 10, 10, 10) (line 47).