**8.16 Grid bag layout manager :-**

When the layout manager is GridBagLayout and a component is within a container, the constraint is an instance of java.awt.GridBagConstraints. Other classes that implement java.awt.LayoutManager2 can have more complex constraints that allow each component more flexibility about its position and size. One of the most powerful of these is java.awt.GridBagLayout.

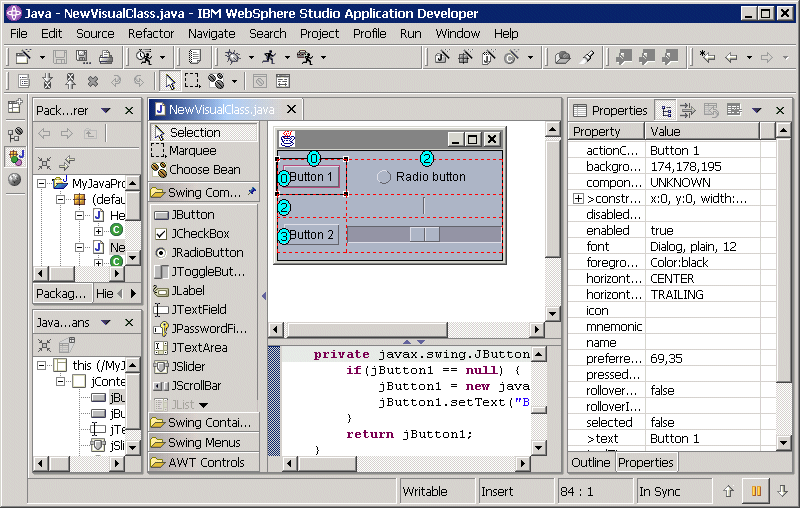
The grid bag layout arranges its components in a row column layout. The GridBagConstraints object contains information about which row and column the component should be placed in, the number of columns the component should span, and how it should be sized and positioned within the column. The constraints property for the GridBagConstraints object lets you expand it and set the individual properties to precisely control its component's placement.

To show a grid for GridBagLayout:

1. In the Design view of a visual class, right-click the container that has a GridBagLayout as its layout manager to open the pop-up menu.
2. Click **Show grid**. (Right-click again to **Hide grid**).

When components are in a container, and you select **Show grid**, red dotted gridlines and values for grids x and y appear as numbers in blue circles. As you click and drag components to different areas, you can easily design your application since you know exactly where each component is on the grid. The numbers across the top of the grid represent the grid x positions and the numbers down the left side of the grid represent the grid y positions. If one or more of the grid positions is skipped, it indicates there are no components with GridBagConstraints that have that specific grid x / grid y setting.

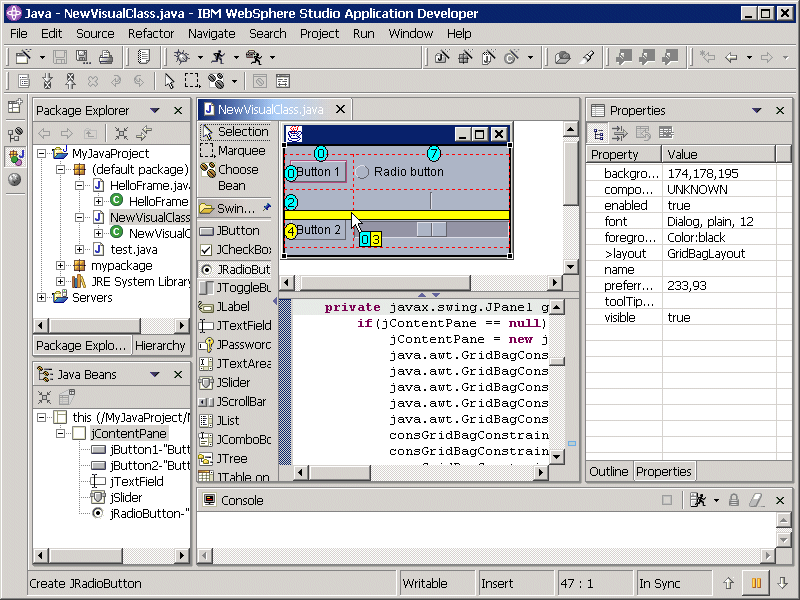
When you click a component in the Design view, the properties values for that component will be reflected in the properties view. For example, in the image below Button 1 is selected, and its values are reflected in the properties view.



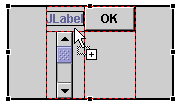
To build your application visually, use the grid to move components. Click and hold the left mouse button on a component, and drag it to another location within the container. If the space is occupied, your cursor will change to a black circle with a line through it, informing you that you cannot move it to that place. If the space is not occupied, then your cursor will remain the same and the empty box will turn to dark grey.

Just below the cursor the x and y grid values for the component you are dropping will appear. The x position specifies the column, y specifies the row. The gridbag starts with 0,0 in the upper left, and moves right and down as x and y increase. If you are dropping or moving a component to an area that is in between two components, that area will change into a vertical or horizontal yellow line, which represents a new column or row, respectively.

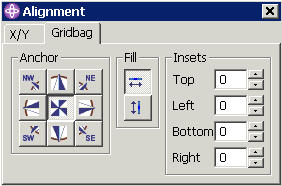
The new x and or y values will appear in yellow circles for the columns and/or rows that are affected by inserting a column and/or row. The grid x and/or grid y values for all components that occupy the cells where the circles are yellow, will be changed to reflect the new grid x and/or grid y values. After the component is dropped onto the canvas, the yellow circles will change back to blue.



Unlike the layout managers FlowLayout, BorderLayout, and null that let you specify a constraint when you drop a component on the parent container, when you drop a component into a GridBagLayout container it is always generated with a default constraint. In addition to dropping into a specific cell location or inserting into a new row and/or column, the Visual Editor for Java™ will set some default information based on the component type in order to display the component better on a GridBagLayout. More specifically, the Visual Editor for Java may set the fill, weight x, and/or weight y values of the GridBagConstraints object based on the component type. For example, when you drop a javax.swing.JTextField, the fill is set to HORIZONTAL (default value is NONE), and the weight x value is set to 1 (default value is 0). Otherwise, the JTextField would be barely visible. In the example below, the JScrollBar was dropped with a fill value set to VERTICAL and the weight y value set to 1.



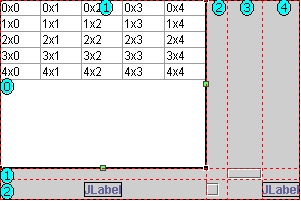
The generated constraint for a new component instantiates GridBagConstraints using its null constructor. To further refine the details such as the grid x and y values, Anchor, fill, and inset properties, you must either expand and edit the constraint value in the Properties view, or go to the toolbar and open the 'Show alignment window' icon. show alignment window icon.



This window shows the anchor, fill, and insets values of the GridBagConstraints for a selected component on the GridBagLayout. If multiple components are selected, the anchor value will only show a pressed button if all the components have the same anchor value. The same is true for the fill value. For insets, the values shown are associated with the last component selected. If you desire to change the value(s) for multiple components, simply select the components on the GridBag in the Design view. Then, in this alignment window select the desired button in the anchor or fill section. All the selected components will be updated at the same time. To change the insets, click on one of the small up or down arrows next to the number field, or type in a value in the field. The value is immediately applied to all components for that position (i.e. Top, Left, Bottom, or Right). Another way to change the insets value is to click into one the fields and hold down the "up arrow" or "down arrow" on the keyboard until the desired value is reached.

When switching to a GridBagLayout from another layout manager, the Visual Editor for Java will generate constraints for each component based on their existing bounds, and the net effect will be that the components do not move their position. Instead of being placed there by an absolute rectangle such as a container with a layout set to null, they have GridBagConstraints.

The advantage of using GridBagConstraints instead of using null layout is that each component is sized according to its preferred size. So, if the size of the parent window changes or if the preferred sizes change at runtime due to different language strings, the layout manager will resize and reposition all of the components and make the best use of the available space.



import java.applet.Applet;

import java.awt.\*;

/\*

<head>

<APPLET CODE="Grid.class" HEIGHT=400 WIDTH=400>

</APPLET>

</body>

\*/

public class Grid extends Applet

{

public void init()

{

// we must explicitly set GridLayout as the manager

setLayout (new GridLayout(5, 3)); // 5 rows, 3 columns, no gaps

for (int row = 0; row < 5; row ++)

{

add (new Label("Label " + row));

add (new Button("Button " + row));

add (new TextField("TextField " + row));

} } }