Introduction

The class **GridBagLayout** arranges components in a horizontal and vertical manner.

Class declaration

Following is the declaration for **java.awt.GridBagLayout** class:

```java
public class GridBagLayout
    extends Object
    implements LayoutManager2, Serializable
```

Field

Following are the fields for **java.awt.BorderLayout** class:

- `double[] columnWeights` -- This field holds the overrides to the column weights.
- `int[] columnWidths` -- This field holds the overrides to the column minimum width.
- `protected Hashtable comptable` -- This hashtable maintains the association between a component and its gridbag constraints.
- `protected GridBagConstraints defaultConstraints` -- This field holds a gridbag constraints instance containing the default values, so if a component does not have gridbag constraints associated with it, then the component will be assigned a copy of the defaultConstraints.
- `protected java.awt.GridBagLayoutInfo layoutInfo` -- This field holds the layout information for the gridbag.
- `protected static int MAXGRIDSIZE` -- The maximum number of grid positions bothhorizontallyandvertically that can be laid out by the grid bag layout.
- `protected static int MINSIZE` -- The smallest grid that can be laid out by the grid bag layout.
- `protected static int PREFERREDSIZE` -- The preferred grid size that can be laid out by the grid bag layout.
- `int[] rowHeights` -- This field holds the overrides to the row minimum heights.
- `double[] rowWeights` -- This field holds the overrides to the row weights.

Class constructors

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Constructor &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GridBagLayout</td>
</tr>
<tr>
<td></td>
<td>Creates a grid bag layout manager.</td>
</tr>
</tbody>
</table>

Class methods

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Method &amp; Description</th>
</tr>
</thead>
</table>
void addLayoutComponent

**Component** component, **Object** constraints

Adds the specified component to the layout, using the specified constraints object.

---

void addLayoutComponent

**String** name, **Component** component

Adds the specified component with the specified name to the layout.

---

protected void adjustForGravity

**GridBagConstraints** constraints, **Rect**angler

Adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.

---

protected void AdjustForGravity

This method is obsolete and supplied for backwards compatibility only; new code should call adjustForGravity instead.

---

protected void arrangeGrid

Lays out the grid.

---

protected void ArrangeGrid

This method is obsolete and supplied for backwards compatibility only; new code should call arrangeGrid instead.

---

**GridBagConstraints** getConstraints

**Component** component

Gets the constraints for the specified component.

---

float getLayoutAlignmentX

**Container** parent

Returns the alignment along the x axis.

---

float getLayoutAlignmentY

**Container** parent

Returns the alignment along the y axis.

---

int[][] getLayoutDimensions

Determines column widths and row heights for the layout grid.

---

protected java.awt.GridBagLayoutInfo getLayoutInfo

**Container** parent, **int** size

Fills in an instance of GridBagLayoutInfo for the current set of managed children.

---

protected java.awt.GridBagLayoutInfo GetLayoutInfo

**Container** parent, **int** size
This method is obsolete and supplied for backwards compatibility only; new code should call getLayoutInfo instead.

13  **Point getLayoutOrigin**
Determine the origin of the layout area, in the graphics coordinate space of the target container.

14  **double[][] getLayoutWeights**
Determine the weights of the layout grid’s columns and rows.

15  **protected Dimension getMinSize**
Figures out the minimum size of the master based on the information from getLayoutInfo.

16  **protected Dimension GetMinSize**
This method is obsolete and supplied for backwards compatibility only; new code should call getMinSize instead.

17  **void invalidateLayout**
Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.

18  **void layoutContainer**
Lays out the specified container using this grid bag layout.

19  **Point location**
Determines which cell in the layout grid contains the point specified by x, y.

20  **protected GridBagConstraints lookupConstraints**
Retrieves the constraints for the specified component.

21  **Dimension maximumLayoutSize**
Returns the maximum dimensions for this layout given the components in the specified target container.

22  **Dimension minimumLayoutSize**
Determines the minimum size of the parent container using this grid bag layout.

23  **Dimension preferredLayoutSize**
Determines the preferred size of the parent container using this grid bag layout.

24

```java
void removeLayoutComponent(Component comp)
```

Removes the specified component from this layout.

25

```java
void setConstraints(Component comp, GridBagConstraints constraints)
```

Sets the constraints for the specified component in this layout.

26

**String toString**

Returns a string representation of this grid bag layout's values.

**Methods inherited**

This class inherits methods from the following classes:

- java.lang.Object

**GridBagLayout Example**

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

**AwtLayoutDemo.java**

```java
package com.tutorialspoint.gui;

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

public class AwtLayoutDemo {
    private Frame mainFrame;
    private Label headerLabel;
    private Label statusLabel;
    private Panel controlPanel;
    private Label msglabel;

    public AwtLayoutDemo(){
        prepareGUI();
    }

    public static void main(String[] args){
        AwtLayoutDemo awtLayoutDemo = new AwtLayoutDemo();
        awtLayoutDemo.showGridBagLayoutDemo();
    }

    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);

msglabel = new Label();
msglabel.setAlignment(Label.CENTER);
msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");

controlPanel = new Panel();
controlPanel.setLayout(new FlowLayout());

mainFrame.add(headerLabel);
mainFrame.add(controlPanel);
mainFrame.add(statusLabel);
mainFrame.setVisible(true);
}

private void showGridBagLayoutDemo(){
    headerLabel.setText("Layout in action: GridBagLayout");

    Panel panel = new Panel();
    panel.setBackground(Color.darkGray);
    panel.setSize(300, 300);
    GridBagLayout layout = new GridBagLayout();

    panel.setLayout(layout);
    GridBagConstraints gbc = new GridBagConstraints();

    gbc.fill = GridBagConstraints.HORIZONTAL;
ge_c.gridx = 0;
ge_c.gridy = 0;
    panel.add(new Button("Button 1"), gbc);

e_c.gridx = 1;
ge_c.gridy = 0;
    panel.add(new Button("Button 2"), gbc);

e_c.fill = GridBagConstraints.HORIZONTAL;
ge_c.ipady = 20;
ge_c.gridx = 0;
ge_c.gridy = 1;
    panel.add(new Button("Button 3"), gbc);

e_c.gridx = 1;
ge_c.gridy = 1;
    panel.add(new Button("Button 4"), gbc);

e_c.gridx = 0;
ge_c.gridy = 2;
ge_c.fill = GridBagConstraints.HORIZONTAL;
ge_c.gridwidth = 2;
    panel.add(new Button("Button 5"), gbc);

    controlPanel.add(panel);

    mainFrame.setVisible(true);
}
Verify the following output

Layout in action: GridBagLayout

Button 1  Button 2

Button 3  Button 4

Button 5

Processing math: 100%