

Layout Managers:-

1.Flow Layout

2.Border Layout

3.Grid Layout

4.Card Layout

5.Grid Bag Layout

Layout Managers:-

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Types of LayoutManager

There are 6 layout managers in Java

- **FlowLayout:** It arranges the components in a container like the words on a page. It fills the top line from **left to right and top to bottom**. The components are arranged in the order as they are added i.e. first components appears at top left, if the container is not wide enough to display all the components, it is wrapped around the line. Vertical and horizontal gap between components can be controlled. The components can be **left, center or right aligned**.
- **BorderLayout:** It arranges all the components along the edges or the middle of the container i.e. **top, bottom, right and left** edges of the area. The components added to the top or bottom gets its preferred height, but its width will be the width of the container and also the components added to the left or right gets its preferred width, but its height will be the remaining height of the container. The components added to the center gets neither its preferred height or width. It covers the remaining area of the container.
- **GridLayout:** It arranges all the components in a grid of **equally sized cells**, adding them from the **left to right and top to bottom**. Only one component can be placed in a cell and each region of the grid will have the same size. When the container is resized, all cells are automatically resized. The order of placing the components in a cell is determined as they were added.
- **GridBagLayout:** It is a powerful layout which arranges all the components in a grid of cells and maintains the aspect ration of the object whenever the container is resized. In this layout, cells may be different in size. It assigns a consistent horizontal and vertical gap among components. It allows us to specify a default alignment for components within the columns or rows.

- **BoxLayout:** It arranges multiple components in either **vertically or horizontally**, but not both. The components are arranged from **left to right or top to bottom**. If the components are aligned **horizontally**, the height of all components will be the same and equal to the largest sized components. If the components are aligned **vertically**, the width of all components will be the same and equal to the largest width components.
- **CardLayout:** It arranges two or more components having the same size. The components are **arranged in a deck**, where all the cards of the same size and the **only top card are visible at any time**. The first component added in the container will be kept at the top of the deck. The default gap at the left, right, top and bottom edges are zero and the card components are displayed either **horizontally or vertically**.

1.FlowLayout:-

- The Java Flow Layout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.
- Flow Layout puts Components(Such as text fields, buttons, labels etc) in a row, if horizontal space is not enough to hold all components then flow layout adds them in a next row and so on.
- All rows in Flow Layout are **center** aligned by default.
- The default horizontal and vertical gap between components is 5 pixels.
- Flow Layout class is present in **java.awt** package.

Class declaration(Syntax):-

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

```
Public class FlowLayout extends Object implements LayoutManager, Serializable
{
-----
-----
}
```

Field or Variable

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

- **static int CENTER** -- This value indicates that each row of components should be centered.
- **static int LEADING** -- This value indicates that each row of components should be justified to the leading edge of the container's orientation, for example, to the left in left-to-right orientations.
- **static int LEFT** -- This value indicates that each row of components should be left-justified.
- **static int RIGHT** -- This value indicates that each row of components should be right-justified.
- **static int TRAILING** -- This value indicates that each row of components should be justified to the trailing edge of the container's orientation, for example, to the right in left-to-right orientations.

Class constructors

S.N.	Constructor & Description
1	FlowLayout() Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.
2	FlowLayout(int align) Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.
3	FlowLayout(int align, int hgap, int vgap)

	Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.
--	--

Class methods

S.N.	Method & Description
1	int getAlignment() Gets the alignment for this layout.
2	int getHgap() Gets the horizontal gap between components.
3	int getVgap() Gets the vertical gap between components.
4	void setAlignment(int align) Sets the alignment for this layout.
5	void setHgap(int hgap) Sets the horizontal gap between components.
6	void setVgap(int vgap) Sets the vertical gap between components.

Methods inherited

This class inherits methods from the following classes:

- java.lang.Object

Example program:-

```
import java.awt.*;
import java.awt.event.*;
public class FlowLayoutExample extends Frame{
    public FlowLayoutExample()
    {

        this.setSize(400,400);

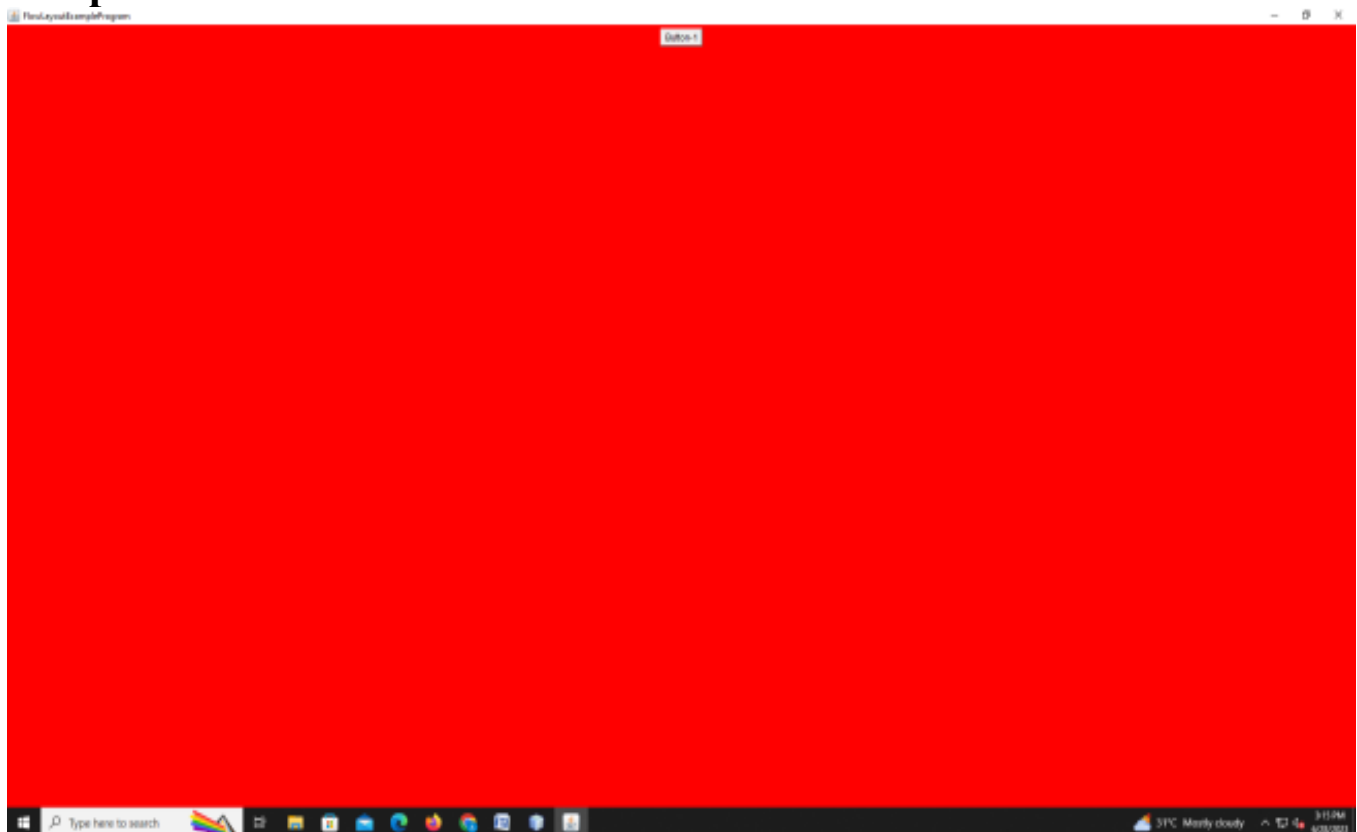
        this.setVisible(true);
        this.setTitle("FlowLayoutExampleProgram");
        this.setBackground(Color.red);
        this.addWindowListener(new WindowAdapter()
```

```
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
Button b1=new Button("Button-1");

this.add(b1);

FlowLayout f=new FlowLayout();
this.setLayout(f);
}
public static void main(String args[])
{
    FlowLayoutExample s=new FlowLayoutExample();
}
}
```

Out put:-



2.BorderLayout:-

- The class **BorderLayout** arranges the components to fit in the five regions: east, west, north, south and center.
- Each region (area) may contain one component only.
- It is the default layout of a frame or window.
- It is present in java.awt package.

Class declaration(Syntax):-

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

Public class BorderLayout extends object implements LayoutManager2, Serializable

```
{  
----  
}
```

Field or Variable:-

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

- **static String NORTH** -- The north layout constraint (top of container).
- **static String SOUTH** -- The south layout constraint (bottom of container).
- **static String WEST** -- The west layout constraint (left side of container).
- **static String EAST** -- The east layout constraint (right side of container).
- **static String CENTER** -- The center layout constraint (middle of container).

Class constructors

S.N.	Constructor & Description
1	BorderLayout() Constructs a new border layout with no gaps between components.
2	BorderLayout(int hgap, int vgap) Constructs a border layout with the specified gaps between components.

Class methods

S.N.	Method & Description
1	void add(Component compObj, Object region) Here, compObj is the component to be added, and region specifies where the component will be added.
2	int getHgap() Gets the horizontal gap between components.
3	int getVgap() Gets the vertical gap between components.

4	void setHgap(int hgap) Sets the horizontal gap between components.
5	void setVgap(int vgap) Sets the vertical gap between components.

Methods inherited

This class inherits methods from the following classes:

- java.lang.Object

Example Program:-

```
import java.awt.*;
import java.awt.event.*;
public class BorderLayoutExample extends Frame{
    public BorderLayoutExample()
    {

        this.setSize(400,400);
        this.setVisible(true);
        this.setTitle("FlowLayoutExampleProgram");
        this.setBackground(Color.red);
        this.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        BorderLayout b=new BorderLayout(20,15);
        this.setLayout(b);
        Button b1=new Button("East");
        Button b2=new Button("West");
        Button b3=new Button("North");
        Button b4=new Button("South");
        Button b5=new Button("Center");

        this.add(b1,"East");
        this.add(b2,"West");
        this.add(b3,"North");
        this.add(b4,"South");
        this.add(b5,"Center");

    }
    public static void main(String args[])
```

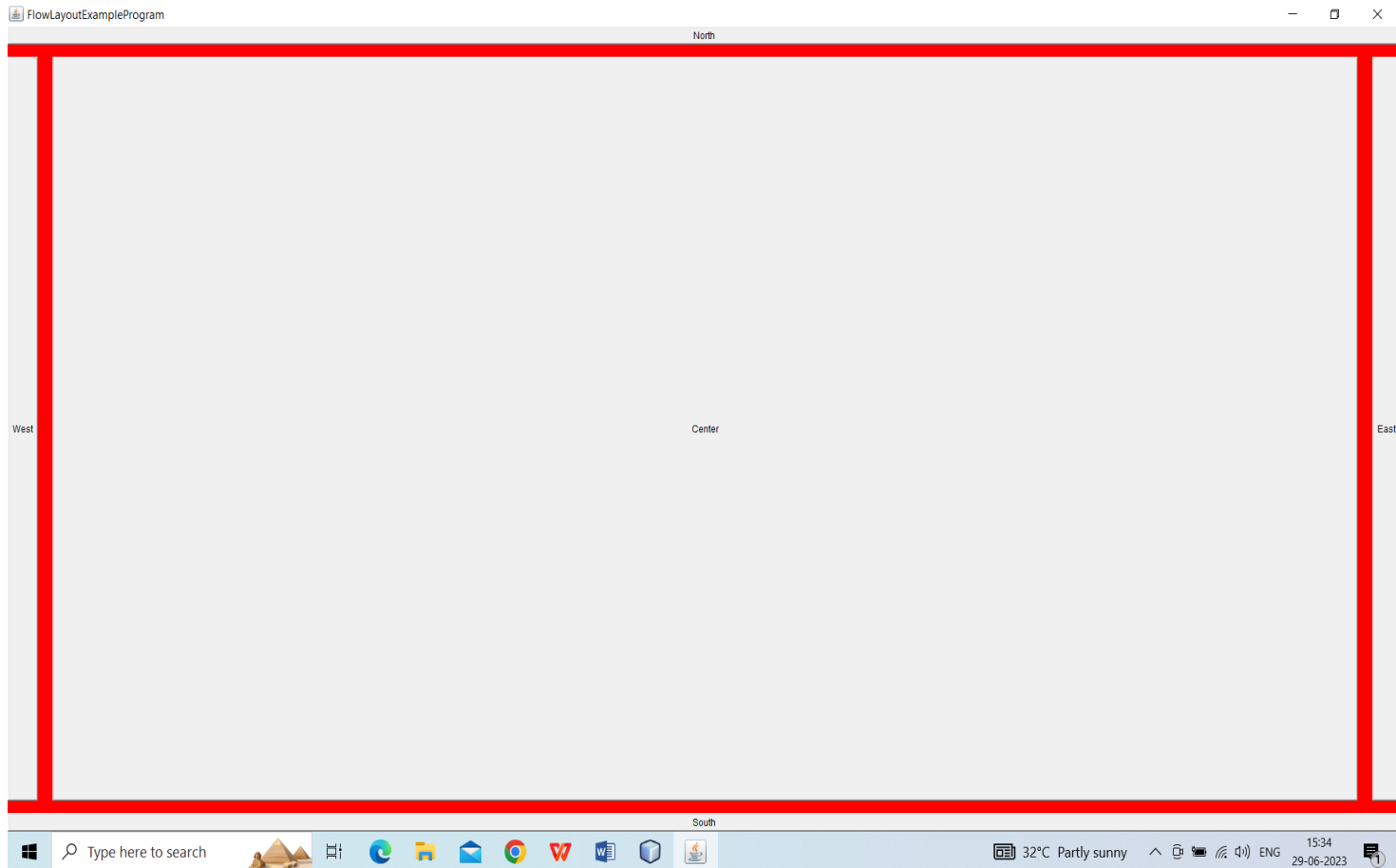
```

{
BorderLayoutExample s=new BorderLayoutExample();
}

}

```

OutPut:-



3.Grid Layout

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle. GridLayout class is present in java.awt package. GridLayout is useful to divide the container into a 2D grid form that contains several rows and columns. The container is divided into equal-sized rectangle; and one component is placed in each rectangle.

Class declaration(Syntax):-

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

```

Public class GridLayout extends Object implements LayoutManager, Serializable
{
-----

-----
}

```

Class Constructors

Sr.No.	Constructor & Description
1	GridLayout() Creates a grid layout with a default of one column per component, in a single row.
2	GridLayout(int rows, int cols) Creates a grid layout with the specified number of rows and columns but no gaps between the components.
3	GridLayout(int rows, int cols, int hgap, int vgap) Creates a grid layout with the specified number of rows and columns along with given horizontal and vertical gaps.

Class Methods

Sr.No.	Method & Description
1	int getColumns() Gets the number of columns in this layout.
2	int getHgap() Gets the horizontal gap between the components.
3	int getRows() Gets the number of rows in this layout.
4	int getVgap() Gets the vertical gap between the components.
5	void setColumns(int cols) Sets the number of columns in this layout to the specified value.
6	void setHgap(int hgap) Sets the horizontal gap between the components to the specified value.

7	void setRows(int rows) Sets the number of rows in this layout to the specified value.
8	void setVgap(int vgap) Sets the vertical gap between the components to the specified value.

Methods Inherited

This class inherits methods from the following class –

- java.lang.Object

Example Program:-

```
import java.awt.*;

import java.awt.event.*;

public class GridLayoutExample extends Frame{

    public GridLayoutExample()

    {

        this.setSize(500,500);

        this.setVisible(true);

        this.setTitle("FlowLayoutExampleProgram");

        this.setBackground(Color.red);

        this.addWindowListener(new WindowAdapter()

        {

            public void windowClosing(WindowEvent e)

            {

                System.exit(0);
```

```
}  
});
```

```
GridLayout g=new GridLayout(3,3);
```

```
this.setLayout(g);
```

```
Button b1=new Button("1");
```

```
Button b2=new Button("2");
```

```
Button b3=new Button("3");
```

```
Button b4=new Button("4");
```

```
Button b5=new Button("5");
```

```
Button b6=new Button("6");
```

```
Button b7=new Button("7");
```

```
Button b8=new Button("8");
```

```
Button b9=new Button("9");
```

```
this.add(b1);
```

```
this.add(b2);
```

```
this.add(b3);
```

```
this.add(b4);
```

```
this.add(b5);
```

```
this.add(b6);
```

```
this.add(b7);
```

```
this.add(b8);
```

```
this.add(b9);
```

```

//b.setHgap(50);

//System.out.println(b.getHgap());

//f.setAlignment(FlowLayout.RIGHT);

}

public static void main(String args[])

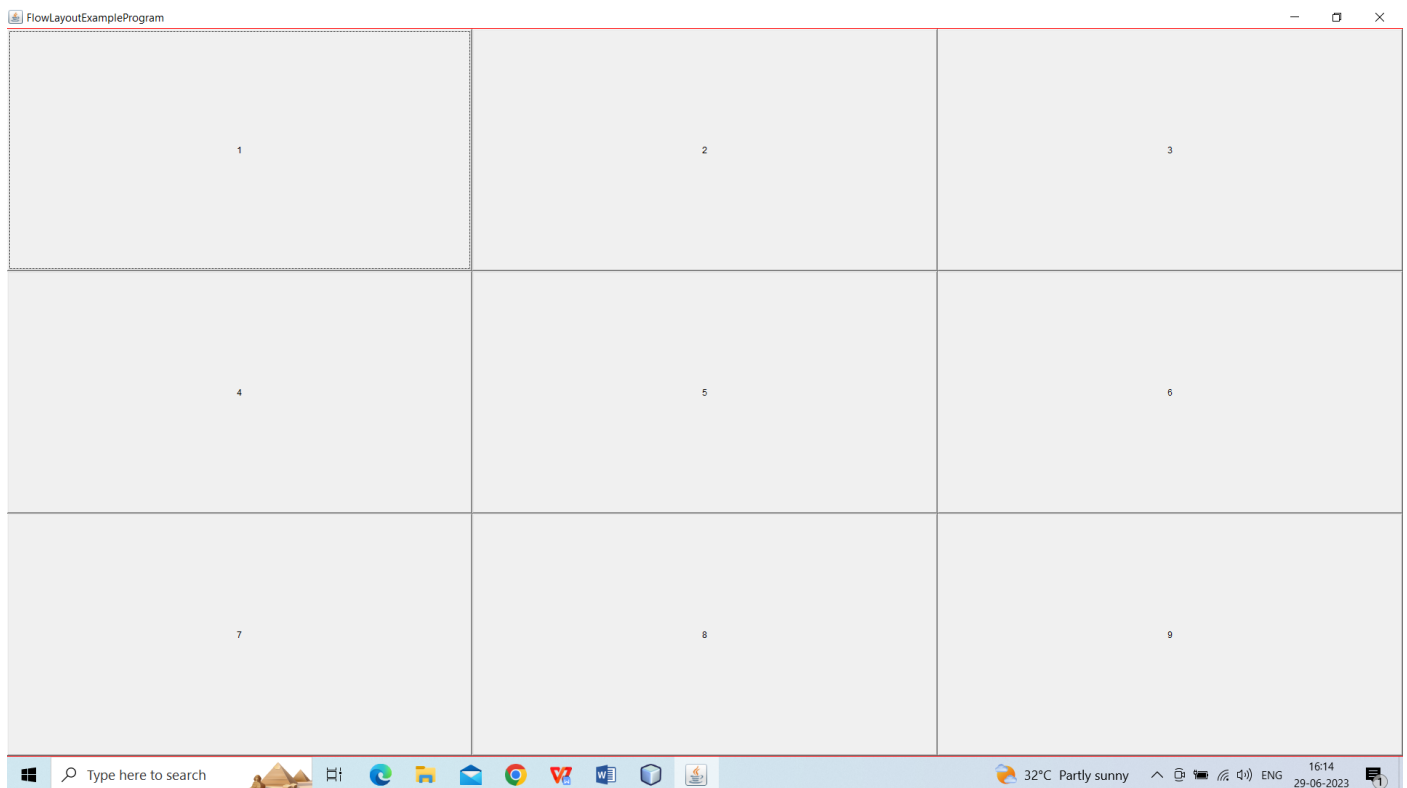
{
GridLayoutExample s=new GridLayoutExample();

}

}

```

OutPut:-



4.Card Layout

- The CardLayout class manages the components in such a way that only one component is visible at a time.
- It treats each component as a card in the container.
- Only one card is visible at a time, and the container acts as a stack of cards.
- The first component added to a CardLayout object is the visible component, when the container is first displayed.
- CardLayout class is found in java.awt package.

Class declaration(Syntax):-

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

Public class CardLayout extends Object implements LayoutManager2, Serializable

```
{  
-----  
  
-----  
}
```

Class Constructors

Sr.No.	Constructor & Description
1	CardLayout() Creates a new card layout with the gaps of size zero.
2	CardLayout(int hgap, int vgap) Creates a new card layout with the specified horizontal and vertical gaps.

Class Methods

Sr.No.	Method & Description
1	void first(Container parent) Flips to the first card of the container.
2	int getHgap() Gets the horizontal gap between the components.

3	int getVgap() Gets the vertical gap between the components.
4	void last(Container parent) Flips to the last card of the container.
5	void next(Container parent) Flips to the next card of the specified container.
6	void previous(Container parent) Flips to the previous card of the specified container.
7	void setHgap(int hgap) Sets the horizontal gap between the components.
8	void setVgap(int vgap) Sets the vertical gap between the components.
9	void show(Container parent, String name) Flips to the component that was added to this layout with the specified name, using addLayoutComponent.

Methods Inherited

This class inherits methods from the following class –

- java.lang.Object

Example:-

```
import java.awt.*;
import java.awt.event.*;
public class CardLayoutExample extends Frame implements ActionListener{
    CardLayout c;
    public CardLayoutExample()
    {

        this.setSize(500,500);

        this.setVisible(true);
    }
}
```

```
this.setTitle("FlowLayoutExampleProgram");
this.setBackground(Color.green);
this.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
});
```

```
c=new CardLayout(30,30);
```

```
this.setLayout(c);
```

```
Button b1=new Button("1");
b1.setBackground(Color.red);
Button b2=new Button("2");
b2.setBackground(Color.green);
Button b3=new Button("3");
b3.setBackground(Color.blue);
Button b4=new Button("4");
b4.setBackground(Color.yellow);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
```

```
this.add(b1);
this.add(b2);
this.add(b3);
this.add(b4);
```

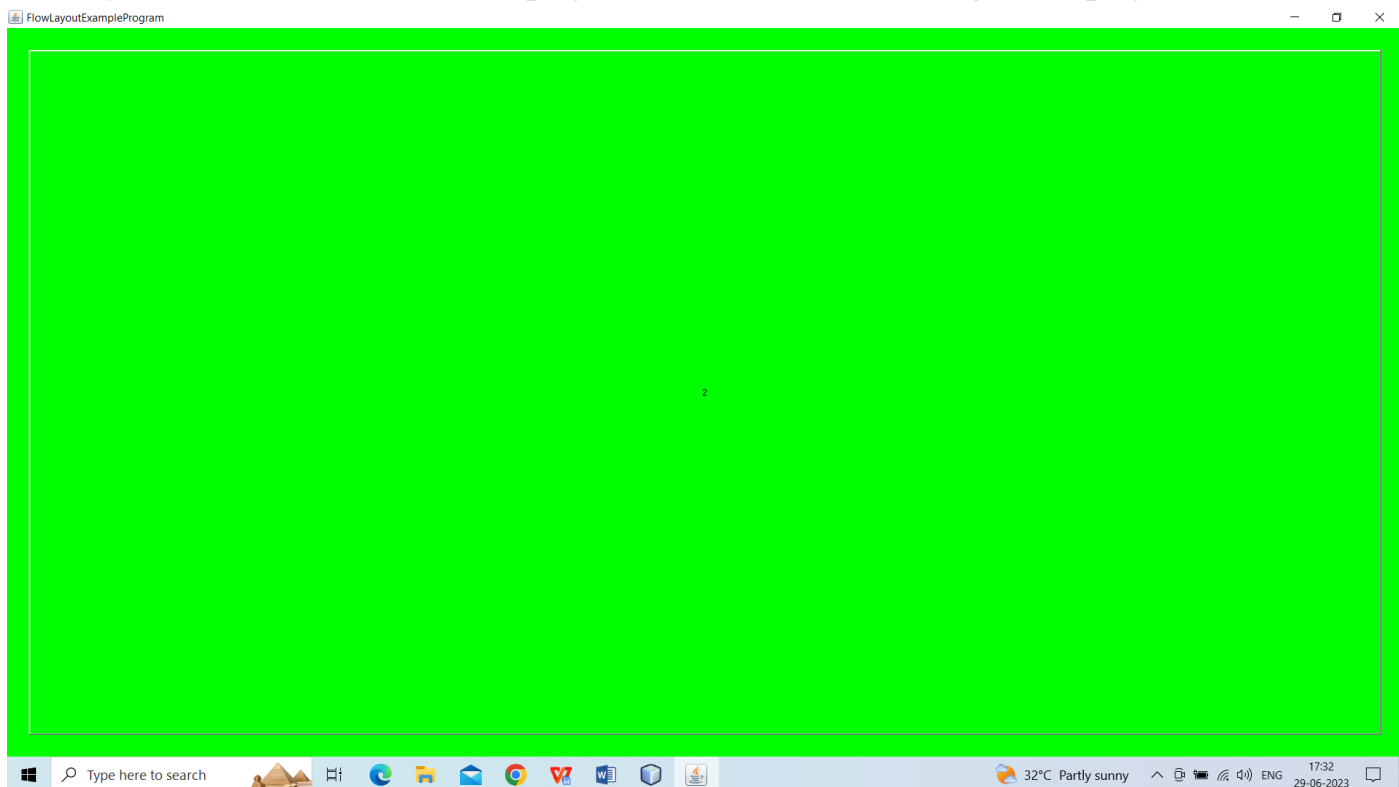
```
}
public void actionPerformed(ActionEvent e)
{
    c.next(this);
}
```

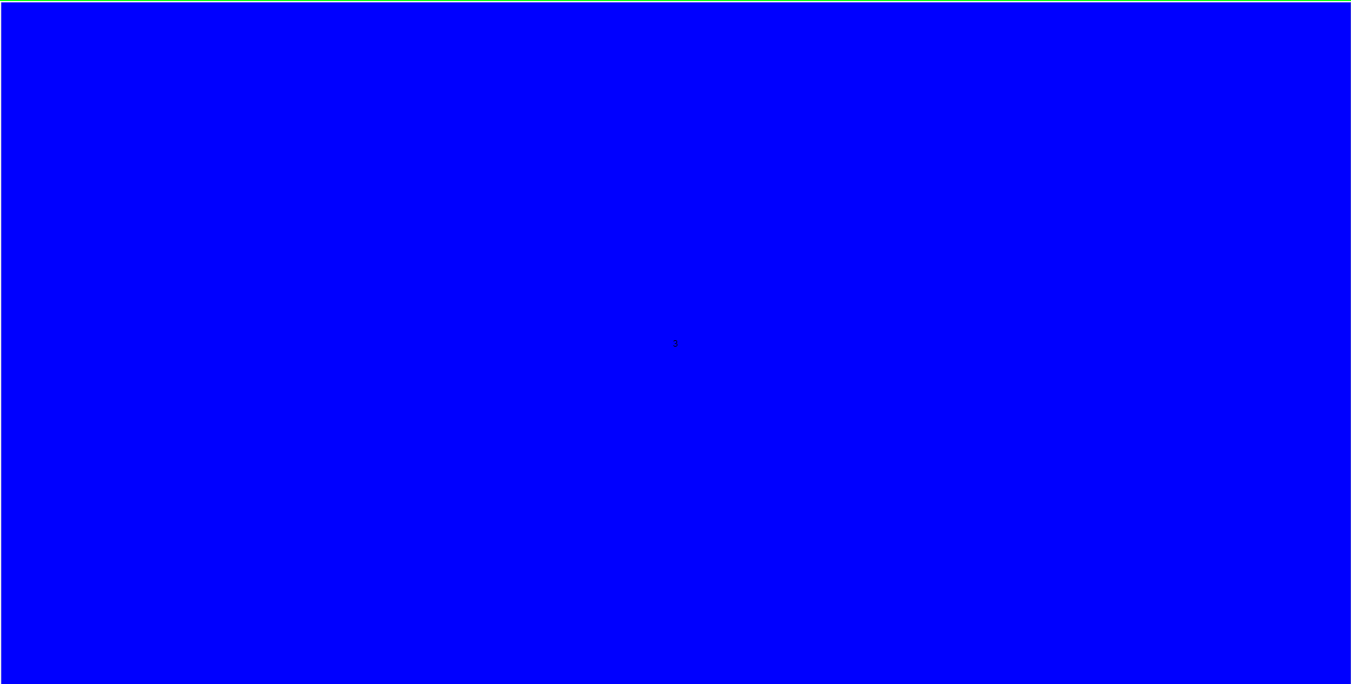
```
public static void main(String args[])
{
    CardLayoutExample s=new CardLayoutExample();
}
```

}
OutPut:-

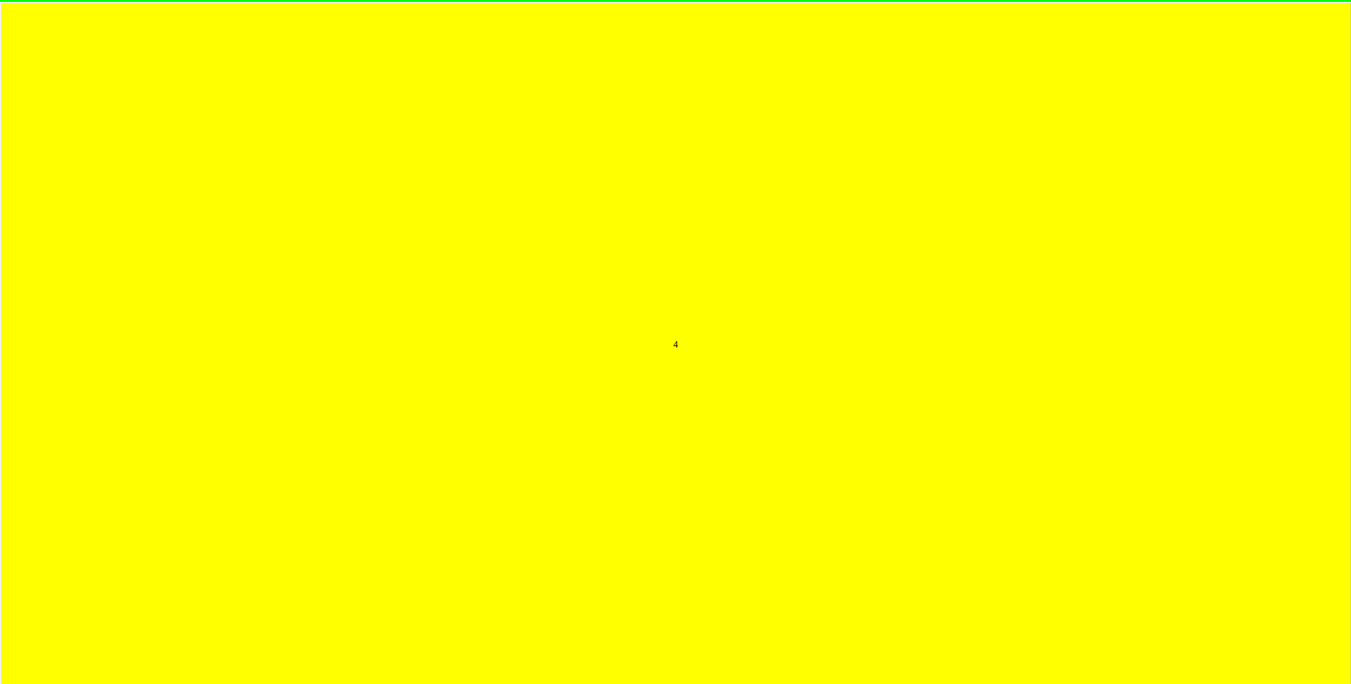


When you click button 1 it will display next button 2 and click again display 3, 4.





3



4

5.Grid Bag Layout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline. A GridBagLayout class represents grid bag layout manager where the components are arranged in rows and columns. In this layout the component can span more than one row or column and the size of the component can be adjusted to fit in the display area.

When positioning the components by using grid bag layout, it is necessary to apply some constraints or conditions on the components regarding their position, size and place in or around the components etc. Such constraints are specified using GridBagConstraints class.

In order to create GridBagLayout, we first instantiate the GridBagLayout class by using its only no-argument constructor

```
GridBagLayout layout=new GridBagLayout();
```

```
setLayout(layout);
```

and defining it as the current layout manager.

To apply constraints on the components, we should first create an object to GridBagConstraints class, as

```
GridBagConstraints gbc =new GridBagConstraints();
```

This will create constraints for the components with default value. The other way to specify the constraints is by directly passing their values while creating the

GridBagConstraints as

```
GridBagConstraints gbc= new GridBagConstraints(
```

```
int gridx, int gridy, int gridwidth, int gridheight, double weightx,
```

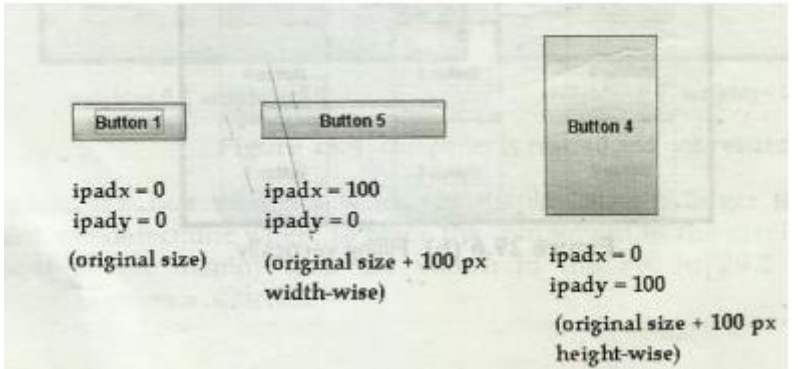
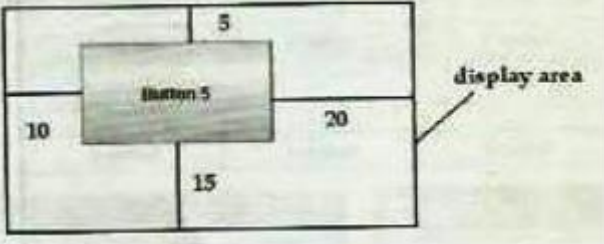
```
double weighty, int anchor, int fill, Insets insets, int ipadx, int ipady );
```

To set the constraints use setConstraints() method in GridBagConstraints class and its prototype

```
void setConstraints(Component comp, GridBagConstraints cons);
```

Constraint fields Defined by GridBagConstraints:

Field	Purpose
int anchor	Specifies the location of a component within a cell. The default is GridBagConstraints.CENTER. Others are <input type="checkbox"/> GridBagConstraints.EAST <input type="checkbox"/> GridBagConstraints.WEST <input type="checkbox"/> GridBagConstraints.SOUTH <input type="checkbox"/> GridBagConstraints.NORTH <input type="checkbox"/> GridBagConstraints.NORTHEAST <input type="checkbox"/> GridBagConstraints.NORTHWEST <input type="checkbox"/> GridBagConstraints.SOUTHEAST <input type="checkbox"/> GridBagConstraints.SOUTHWEST
int gridx	Specifies the X coordinate of the cell to which the component will be added.
int gridy	Specifies the Y coordinate of the cell to which the component will be added.
int gridheight	Specifies the height of component in terms of cells. The default is 1.
int gridwidth	Specifies the width of component in terms of cells. The default is 1.
double weightx	Specifies a weight value that determines the horizontal spacing between cells and the edges of the container that holds them. The default value is 0.0. The greater the weight, the more space that is allocated.
double weighty	Specifies a weight value that determines the vertical spacing between cells and the edges of the container that holds them. The default value is 0.0.
int ipadx	Specifies extra horizontal space that surrounds a component within a cell. The default is 0.

	
int ipady	Specifies extra vertical space that surrounds a component within a cell. The default is 0.
int fill	<p>Specifies how a component is resized if the component is smaller than its cell. Valid values are</p> <ul style="list-style-type: none"> <input type="checkbox"/> GridBagConstraints.NONE (the default) <input type="checkbox"/> GridBagConstraints.HORIZONTAL <input type="checkbox"/> GridBagConstraints.VERTICAL <input type="checkbox"/> GridBagConstraints.BOTH.
Insets insets	<p>Small amount of space between the container that holds your components and the window that contains it. Default insets are all zero.</p> <p>Insets(int top,int left,int bottom, int right)</p> <p>Ex. Insets i=new Insets(5,10,15,20);</p> 

Example:

```
// Use GridBagLayout.
```

```
import java.awt.*;
```

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

```
class MyFrame extends Frame
```

```
{
```

```
Button b1,b2,b3,b4,b5,b6,b7,b8,b9;
```

```
MyFrame()

{
GridBagLayout gbag = new GridBagLayout();
GridBagConstraints gbc = new GridBagConstraints();
setLayout(gbag);

b1=new Button("Button 1");
b2=new Button("Button 2");
b3=new Button("Button 3");
b4=new Button("Button 4");
b5=new Button("Button 5");
b6=new Button("Button 6");
b7=new Button("Button 7");
b8=new Button("Button 8");
b9=new Button("Button 9");

gbc.gridx=0;
gbc.gridy=0;
gbag.setConstraints(b1,gbc);
gbc.gridx=1;
gbc.gridy=0;
gbag.setConstraints(b2,gbc);
gbc.gridx=2;
gbc.gridy=0;
gbag.setConstraints(b3,gbc);
gbc.gridx=0;
gbc.gridy=1;
gbag.setConstraints(b4,gbc);
gbc.gridx=1;
```

```
gbc.gridy=1;
gbag.setConstraints(b5, gbc);
gbc.gridx=2;
gbc.gridy=1;
gbag.setConstraints(b6, gbc);
gbc.gridx=0;
gbc.gridy=2;
gbag.setConstraints(b7, gbc);
gbc.gridx=1;
gbc.gridy=2;
gbag.setConstraints(b8, gbc);
gbc.gridx=2;
gbc.gridy=2;
gbag.setConstraints(b9, gbc);
add(b1);
add(b2);
add(b3);
add(b4);
add(b5);
add(b6);
add(b7);
add(b8);
add(b9);
}
}
class GridBagDemo2
{
```

```

public static void main(String arg[])
{
    MyFrame f=new MyFrame();
    f.setSize(400,400);
    f.setTitle("GridBagLayout Example...");
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent we)
        {
            System.exit(0);
        }
    });
}
}

```

OutPut:-

Event Handling Programs...

— □ ×

Button 1	Button 2	Button 3
Button 4	Button 5	Button 6
Button 7	Button 8	Button 9