Chapter-2.2
Layout Manager

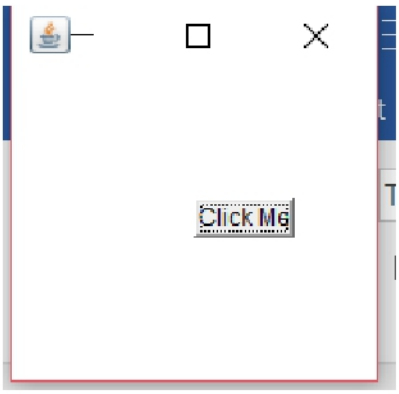
Disabling the Layout Manager / no layout manager

- If the layout is set to null, then components must be sized and positioned by hand using the following methods.

```
component.setSize(width, height)
component.setLocation(left, top)
        or
component.setBounds(left, top,width, height)
```

Example:

```
import java.awt.*;
public class GroupExample extends Frame
{
        Button b1;
        public GroupExample()
        {
                setSize(200,200);
                setLayout(null);//no layout manager
                b1=new Button("Click Me");
                b1.setBounds(100,100,50,20);
                add(b1);
                setVisible(true);
        }
        public static void main(String [] args)
        {
                new GroupExample();
        }
}
```



Layout Manager and Layout management

- A layout manager is an object that determines the size and position of the components within a container.
- A layout manager controls how GUI components are organized within a GUI container.
- Each AWT container (e.g. Frame, Dialog, Panel) and Swing container (e.g. JFrame, JDialog, JPanel) is a subclass of java.awt.Container and so has a layout manager that controls it.
- Layout management is a process to determine the size and position of the components in the container. Some of the layout manager are.
    1. Flow Layout
    2. Border Layout
    3. Grid Layout
    4. Group Layout
    5. GridBag Layout
- We use setLayout() method to specify the type of layout to be used.

1. Flow Layout Manager

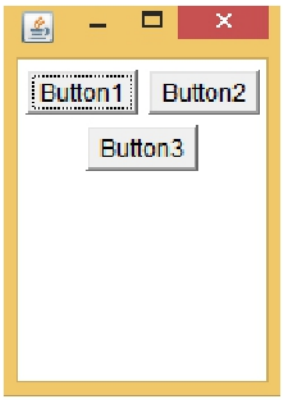
- The simplest layout manager is java.awt.FlowLayout, which adds components to the container from left-to-right, top-to-bottom.
- **It is the default layout for GUI container objects of classes Applet and Panel.**

- It will arrange components such as buttons left to right until no more components(e.g. Button) fit on the same line.
- The constructor used to create Flow Layout are
    i. public FlowLayout(): creates default layout with centrally aligned components and 5 pixels gap in both horizontal and vertical direction.
    ii. public FlowLayout(int alignment): creates a layout with the indicated alignment left(FlowLayout.LEADING), right(FlowLayout.Trailing) or center(FlowLayout.CENTER) with default horizontal and vertical gap.
    iii. public FlowLayout(int alignment, int horizontal_gap, int vertical_gap): creates a layout with indicated alignment with specified horizontal gap and vertical gap.

Example1:

```
import java.awt.*;
public class FlowDemo extends Frame
{
        Button b1,b2,b3;
        public FlowDemo()
        {
                setSize(100,200);
                setLayout(new FlowLayout());
                b1=new Button("Button1");
                b2=new Button("Button2");
                b3=new Button("Button3");
                add(b1);
                add(b2);
                add(b3);
                show();
        }
        public static void main(String [] args)
        {
                new FlowDemo();
        }
}
```

Output:



2. Boarder Layout Manager

- Another simple layout manager is java.awt.BorderLayout, which is the **default layout for GUI container objects of classes Frame and Dialog.**
- Border layout splits the GUI container into five regions – **east, west, north, south and center** –oriented so that north is the top, south is bottom, left is west and right is east.
- In the call to the add method, you must specify to which area of the container the component is to be added, otherwise it is added to the **center area by default**.
- The constructor used to create Boarder Layout are
    i. public BoarderLayout(): creates boarder layout with default vertical and horizontal gaps of zero pixel each.
    ii. public BoarderLayout(int hgap, int vgap):creates a boarder layout with the specified horizontal and vertical gap.
- We use add() method to add the components to the container. The syntax for add () method is

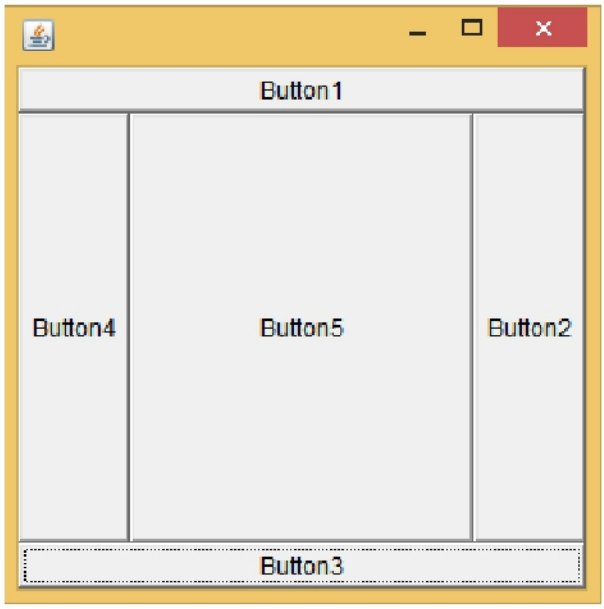
    Add(Component obj, Object reg);

    Where, obj is the components to be added and reg is the region in which components is to be added.

Example:

```
import java.awt.*;
public class BorderDemo extends Frame
{
        Button b1,b2,b3,b4,b5;
        public BorderDemo()
        {
                setSize(300,300);
                setLayout(new BorderLayout());
                b1=new Button("Button1");
                b2=new Button("Button2");
                b3=new Button("Button3");
                b4=new Button("Button4");
                b5=new Button("Button5");
                add(b1,BorderLayout.NORTH);
                add(b2,BorderLayout.EAST);
                add(b3,BorderLayout.SOUTH);
                add(b4,BorderLayout.WEST);
                add(b5,BorderLayout.CENTER);
                show();
        }
        public static void main(String [] args)
        {
                new BorderDemo();
        }
}
```

Output:



3. Grid Layout Manager

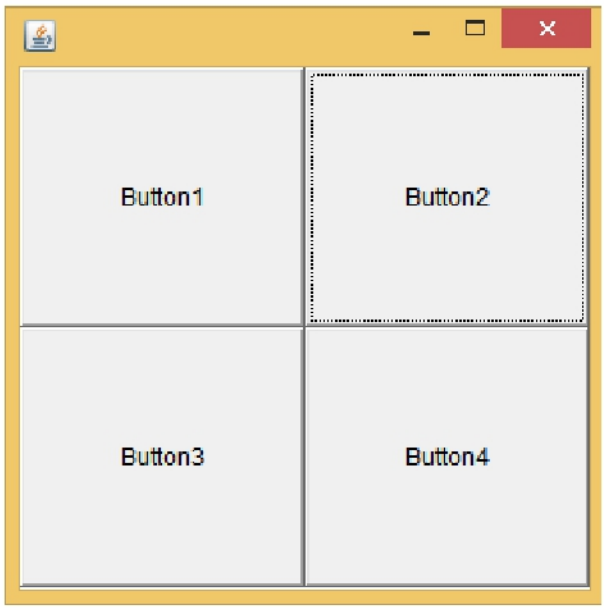
- The third simple layout manager is **java.awt.GridLayout**, which adds components to a rectangular grid in rows – columns order.
- The container is divided into equal sized rectangles, and one component is placed in each rectangle.
- The constructors used to create Grid Layout are
  - i. public GridLayout(): creates a grid layout with a default of one column per component in a single row.
  - ii. public GridLayout(int rows, int colmns): creates a grid layout with the specified number of rows and columns

iii. public GridLayout(int rows, int colmns, int hgap, int vgap): creates a grid layout with specified number of rows and columns such that hgap is spacing between columns and vgap is spacing between rows in pixels.

Example1:

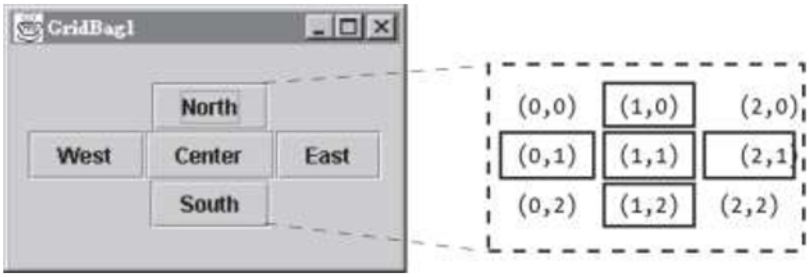
```
import java.awt.*;
public class GridDemo extends Frame
{
        Button b1,b2,b3,b4;
        public GridDemo()
        {
                setSize(300,300);
                setLayout(new GridLayout(2,2));
                b1=new Button("Button1");
                b2=new Button("Button2");
                b3=new Button("Button3");
                b4=new Button("Button4");
                add(b1);
                add(b2);
                add(b3);
                add(b4);
                show();
        }
        public static void main(String [] args)
        {
                new GridDemo();
        }
}
```

Output:



Grid Bag Layout Manager

- **java.awt.GridBagLayout** allows very sophisticated placement of components within a container.
- It's essentially a grid, but the grid rows and columns may have varying heights and widths and components may span more than one grid cell.



- Consequently, it's a rather challenging layout manager to learn and use.
- It divides the window into grids, without requiring the components to be the same size.
- It is more flexible than the other standard layout managers, but harder to use.
- Each component managed by a grid bag layout is associated with an instance of GridBagConstraints
- The GridBagConstraints specifies:
  - How the component is laid out in the display area
  - In which cell the component starts and ends
  - How the component stretches when extra room is available
  - Alignment in cells
- The constructor used to create Grid Bag Layout is

  public GridBagLayout()

  Example: GridBagLayout lay=new GridBagLayout();

- The constructor uses to create Grid Bag Consttraints is

  public GridBagConstraints()

  Example: GridBagConstraints constraints=new GridBagConstraints();

Some of the fields/instance variables of GridBabConstraints are

i. gridx and gridy

gridx, gridy specifies the top-left corner of the component. Upper left of grid is located at (gridx, gridy)=(0,0)

ii. fill

Specifies what to do to an element that is smaller than the cell size

constraints.fill = GridBagConstraints.HORIZONTAL;

The size of row/column is determined by the widest/tallest element in it

Can be NONE, HORIZONTAL, VERTICAL, or BOTH

iii. anchor:

Specifies where the component should be positioned in its display area:

constraints.anchor = GridBagConstraints.NORTHEAST;

Note: the fill must be set to GridBagConstraints.NONE to use anchor.

Can be NORTH, EAST, SOUTH, WEST, NORTHEAST,NORTHWEST, SOUTHEAST,SOUTHWEST or CENTER

iv. gridheight and gridwidth

Specifies the number of rows and columns the Component occupies

constraints.gridwidth = 3; i.e.component will occupy 3 columns

constraints.gridheight=3; i.e. components will occupy 3 rows

Note: **GridBagConstraints.REMAINDER** lets the component take up the remainder of the row/column

v. insets: The minimum amount of space between a component and the edge of its display area.

cons.insets=new Insets(int top_space,int left_space,int bottom_space,int right_space);

Note:

- setConstraints() method will set the constraints not only just before it but all the constraints defined from top to the constraint defined above this statement.
- Componetnts can be added either using

  add(Component,GridBagConstraints) method

  or

  lay.setConstraints(Component, GridBagConstraints)

  followed by method

  add(Component)*

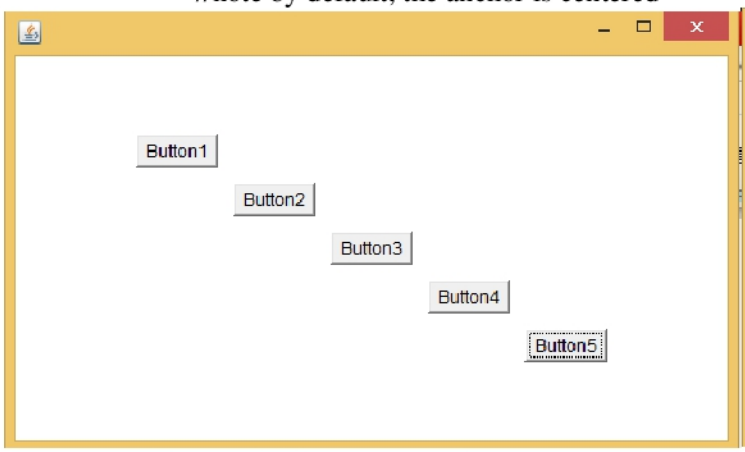
Program1:

```
import java.awt.*;
public class GridBagDemo extends Frame
{
        GridBagConstraints cons;
        Button b1,b2,b3,b4,b5;
        public GridBagDemo()
```

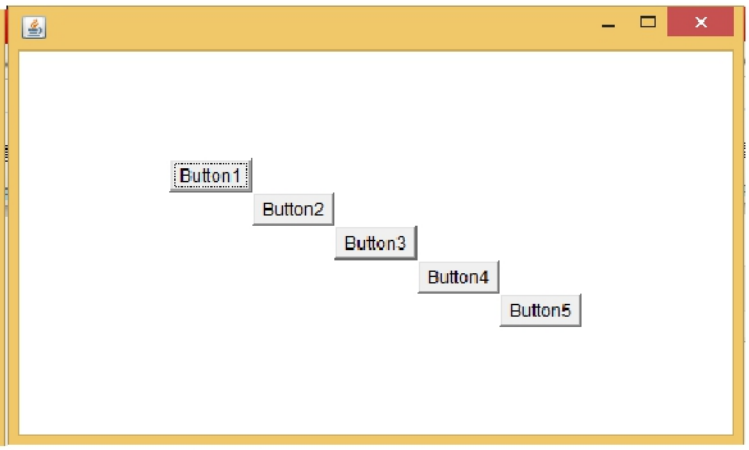
```
        {
                cons=new GridBagConstraints();
                cons.insets=new Insets(5,5,5,5); (Check the output with and without this constraints)
                b1=new Button("Button1");
                b2=new Button("Button2");
                b3=new Button("Button3");
                b4=new Button("Button4");
                b5=new Button("Button5");
                setSize(500,300);
                setLayout(new GridBagLayout());

                cons.gridx=0;
                cons.gridy=0;
                add(b1,cons);
                cons.gridx=1;
                cons.gridy=1;
                add(b2,cons);
                cons.gridx=2;
                cons.gridy=2;
                add(b3,cons);
                cons.gridx=3;
                cons.gridy=3;
                add(b4,cons);
                cons.gridx=4;
                cons.gridy=4;
                add(b5,cons);

                show();
        }
        public static void main(String [] args)
        {
                new GridBagDemo();
        }
}
//note by default, the anchor is centered
```



Output with insets constraints



output without insets constraints

Program2:

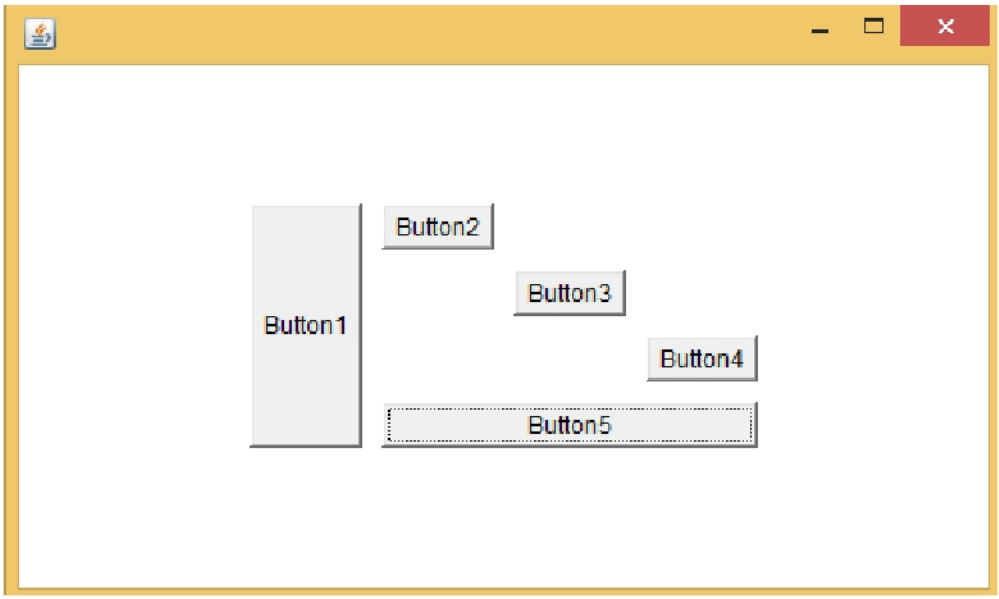
```
import java.awt.*;
public class GridBagDemo extends Frame
{
        GridBagLayout glayout;
        GridBagConstraints cons;
        Button b1,b2,b3,b4,b5;
        public GridBagDemo()
        {
                cons=new GridBagConstraints();
                b1=new Button("Button1");
```

```
                b2=new Button("Button2");
                b3=new Button("Button3");
                b4=new Button("Button4");
                b5=new Button("Button5");
                setSize(500,300);
                setLayout(new GridBagLayout());
                cons.insets=new Insets(5,5,5,5);

                cons.gridx=0;
                cons.gridy=0;
                cons.gridheight=5;
                cons.fill=GridBagConstraints.VERTICAL;
                add(b1,cons);
                cons.gridx=1;
                cons.gridy=0;
                cons.gridheight=1;
                add(b2,cons);
                cons.gridx=2;
                cons.gridy=1;
                add(b3,cons);
                cons.gridx=3;
                cons.gridy=2;
                add(b4,cons);
                cons.gridx=1;
                cons.gridy=3;
                cons.gridwidth=GridBagConstraints.REMAINDER;// will return 3 columns because remainder means
                                                            // remaining space after
                cons.fill=GridBagConstraints.HORIZONTAL;// for stretching button5
                 add(b5,cons);

                show();
        }


        public static void main(String [] args)
        {
                new GridBagDemo();
        }
}
```



4. Group Layout Manager

- GroupLayout groups its components and places them in a Container hierarchically. The grouping is done by instances of the Group class.
- GroupLayout treats each the horizontal and vertical layouts separately. That is, there is a group representing the horizontal axis, and a group representing the vertical axis. Each component must exist in both a horizontal and vertical group, otherwise an IllegalStateException is thrown.
- Mostly used by IDE like netbeans as default layout manager for JFrame.
- To use GroupLayout we must import **javax.swing** package.
- GroupLayout uses two types of arrangements
  - i. Sequential arrangement
    - With sequential arrangement, the components are simply placed one after another, just like FlowLayout would do along one axis.
    - The position of each component is defined as being relative to the preceding component.
  - ii. Parallel arrangement
    - The second way places the components in parallel—on top of each other in the same space.
    - They can be baseline-, top-, or bottom-aligned along the vertical axis.
    - Along the horizontal axis, they can be left-, right-, or center-aligned if the components are not all the same size.

Example:

- Let C1, C2 and C3 be three different components.

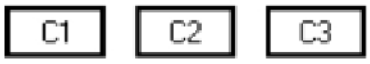
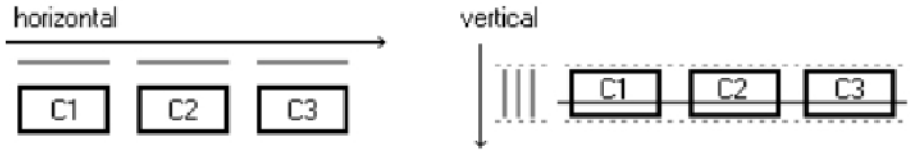


Fig: Three components in a row

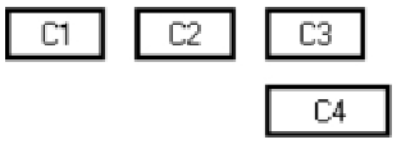
- Expressing this layout using groups. Starting with the horizontal axis, there is a sequential group of 3 components arranged from left to right. Along the vertical axis there is a parallel group of the same 3 components with the same location, size, and baseline.



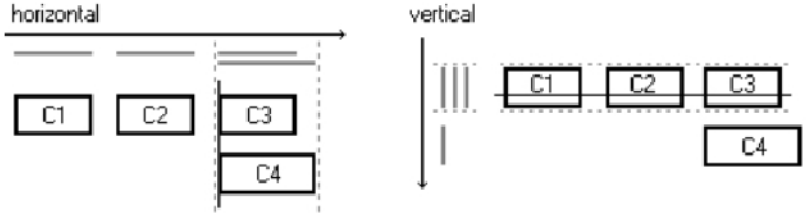
In pseudo code, the layout specification might look like this

horizontal layout = sequential group { c1, c2, c3 }
vertical layout = parallel group (BASELINE) { c1, c2, c3 }

- So we clearly see that components grouped sequentially in one dimension usually form a parallel group in the other dimension.
- Now let us add one more component, C4, left-aligned with C3 as below.



- Along the horizontal axis the new component occupies the same horizontal space as C3 so that it forms a parallel group with C3. Along the vertical axis C4 forms a sequential group with the original parallel group of the three components.



In pseudo code, the layout specification might look like this

horizontal layout = sequential group { c1, c2, parallel group (LEFT) { c3, c4 } }

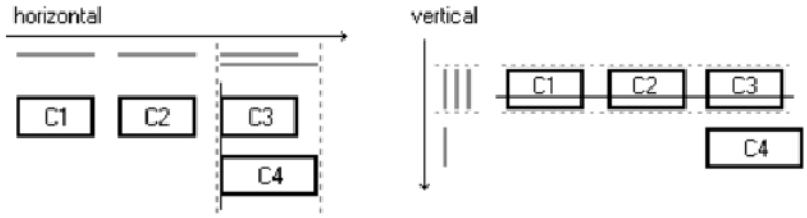
vertical layout = sequential group { parallel group (BASELINE) { c1, c2, c3 }, c4 }

- The constructors used to create Group Layout is

  public GroupLayout(Container host)

- Some commonly used methods are
  1. createSequentialGroup(): creates and returns a SequentialGroup
  2. createParallelGroup(): It creates and returns a ParallelGroup with an alignment of Alignment.LEADING
  3. addComponent(Component): it adds component to a group
  4. addGroup(GroupLayout.ParallelGroup): It is used to add parallel group to an existing sequential group.

Example




```
import javax.swing.*;
import java.awt.*;
public class GroupExample extends Frame
{
        Button b1,b2,b3,b4;
        public GroupExample()
        {
                setSize(1000,200);
                GroupLayout g=new GroupLayout(this);
                b1=new Button("First Button");
                b2=new Button("Second Button");
                b3=new Button("Third Button");
                b4=new Button("Fourth Button");
                setLayout(g);
                g.setAutoCreateGaps(true);   //auto gaps between components
                g.setAutoCreateContainerGaps(true);   //auto gap between components and container boundary

                GroupLayout.SequentialGroup hGroup = g.createSequentialGroup();   //horizontal group
                hGroup.addComponent(b1);
                hGroup.addComponent(b2);

                //b3 and b4 are in parallal group in horizontal axis
                GroupLayout.ParallelGroup hpgroup=g.createParallelGroup(GroupLayout.Alignment.LEADING);
                hpgroup.addComponent(b3);
                hpgroup.addComponent(b4);

                hGroup.addGroup(hpgroup);

                GroupLayout.SequentialGroup vGroup = g.createSequentialGroup(); // vertical group
                GroupLayout.ParallelGroup vpgroup=g.createParallelGroup();
                vpgroup.addComponent(b1);
                vpgroup.addComponent(b2);
                vpgroup.addComponent(b3);
                vGroup.addGroup(vpgroup);
                vGroup.addComponent(b4);

                g.setHorizontalGroup(hGroup);
                g.setVerticalGroup(vGroup);

                setVisible(true);
```

```
        }
        public static void main(String [] args)
        {
                new GroupExample();
        }
}
```

Create a GUI frame to find addition of two number using

- **i. flowlayout manager**
- **ii. GridLayout**
- **iii. GridBagLayout manager.**
- **iv. GroupLayout manager**

1. Using FlowLayout Manager

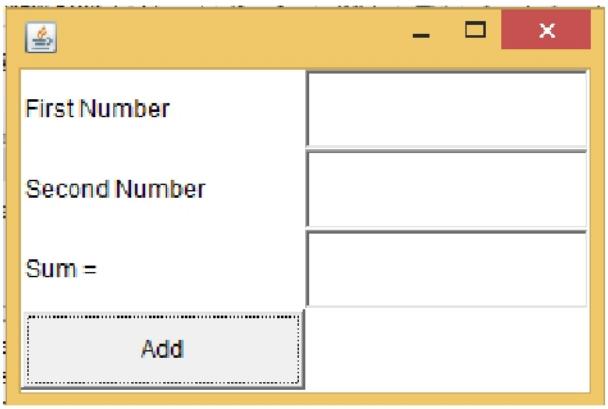
```
import java.awt.*;
public class Demo extends Frame
{
        Label lbl_fn,lbl_sn,lbl_res;
        Button btn_add;
        TextField txt_fn,txt_sn,txt_res;

        public Demo()
        {
                setSize(1000,200);
                setLayout(new FlowLayout());
                lbl_fn=new Label("First Number");
                add(lbl_fn);
                txt_fn=new TextField(20);
                add(txt_fn);
                lbl_sn=new Label("Second Number");
                add(lbl_sn);
                txt_sn=new TextField(20);
                add(txt_sn);
                lbl_res=new Label("Sum =");
                add(lbl_res);
                txt_res=new TextField(20);
                add(txt_res);
                btn_add=new Button("Add");
                add(btn_add);
                setVisible(true);
        }
        public static void main(String [] args)
        {
                new Demo();
        }
}
```

2. **Using GridLayout**

```
import java.awt.*;
public class AddDemo1 extends Frame
{
    GridLayout glayout;
    Label lbl_fn,lbl_sn,lbl_res;
    Button btn_add;
    TextField txt_fn,txt_sn,txt_res;
    public AddDemo1()
    {
        setSize(300,200);
        glayout=new GridLayout(4,2);
        setLayout(glayout);
        lbl_fn=new Label("First Number");
        add(lbl_fn);
        txt_fn=new TextField(20);
        add(txt_fn);
        lbl_sn=new Label("Second Number");
        add(lbl_sn);
        txt_sn=new TextField(20);
        add(txt_sn);
        lbl_res=new Label("Sum =");
        add(lbl_res);
        txt_res=new TextField(20);
        add(txt_res);
        btn_add=new Button("Add");
        add(btn_add);
        show();
    }
    public static void main(String [] args)
    {
        new AddDemo1();
    }
}
```

Output:



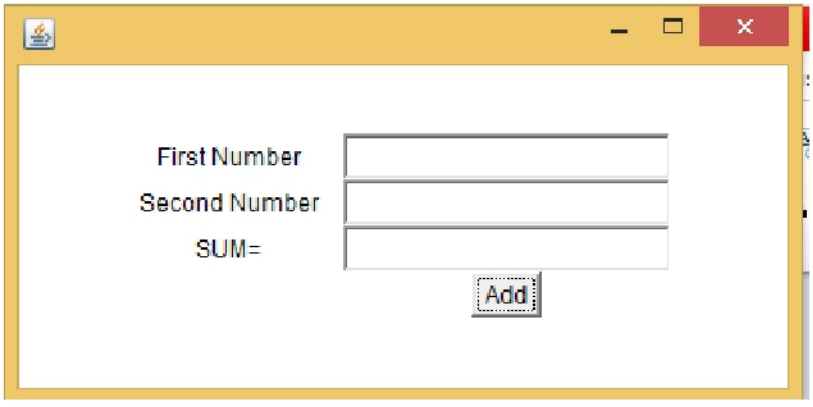
3. **Using GridBagLayout Manager**

```
import java.awt.*;
public class AddDemo extends Frame
{
    GridBagLayout glayout;
    GridBagConstraints cons;
```

```
        Label lbl_fn,lbl_sn, lbl_res;
        TextField txt_fn,txt_sn,txt_res;
        Button btn_add;
        public AddDemo()
        {
                setSize(400,200);
                glayout=new GridBagLayout();
                cons=new GridBagConstraints();
                setLayout(glayout);
                lbl_fn=new Label("First Number");
                cons.gridx=0;
                cons.gridy=0;
                add(lbl_fn,cons);
                cons.gridx=1;
                cons.gridy=0;
                txt_fn=new TextField(20);
                add(txt_fn,cons);
                lbl_sn=new Label("Second Number");
                cons.gridx=0;
                cons.gridy=1;
                add(lbl_sn,cons);
                txt_sn=new TextField(20);
                cons.gridx=1;
                cons.gridy=1;
                add(txt_sn,cons);
                lbl_res=new Label("SUM=");
                cons.gridx=0;
                cons.gridy=2;
                add(lbl_res,cons);
                txt_res=new TextField(20);
                cons.gridx=1;
                cons.gridy=2;
                add(txt_res,cons);
                btn_add=new Button("Add");
                cons.gridx=1;  // 0 also works
                cons.gridy=3;
                add(btn_add,cons);
                show();
        }

        public static void main(String [] args)
        {
                new AddDemo();
        }
}
```

Output:



4. Using GroupLayout manager

```
import javax.swing.*;
import java.awt.*;
public class Demo extends Frame
{
	Label lbl_fn,lbl_sn,lbl_res;
	Button btn_add;
	TextField txt_fn,txt_sn,txt_res;

	public Demo()
	{
		setSize(1000,200);
		lbl_fn=new Label("First Number");
		txt_fn=new TextField(20);
		lbl_sn=new Label("Second Number");
		txt_sn=new TextField(20);
		lbl_res=new Label("Sum =");
		txt_res=new TextField(20);
		btn_add=new Button("Add");

		GroupLayout g=new GroupLayout(this);
		setLayout(g);

		GroupLayout.SequentialGroup hGroup = g.createSequentialGroup();
		GroupLayout.ParallelGroup hpgroup1=g.createParallelGroup();
		hpgroup1.addComponent(lbl_fn);
		hpgroup1.addComponent(lbl_sn);
		hpgroup1.addComponent(lbl_res);
		hpgroup1.addComponent(btn_add);
		hGroup.addGroup(hpgroup1);

		GroupLayout.ParallelGroup hpgroup2=g.createParallelGroup();
		hpgroup2.addComponent(txt_fn);
		hpgroup2.addComponent(txt_sn);
		hpgroup2.addComponent(txt_res);
		hGroup.addGroup(hpgroup2);


		GroupLayout.SequentialGroup vGroup = g.createSequentialGroup();
		GroupLayout.ParallelGroup vpgroup1=g.createParallelGroup();
		vpgroup1.addComponent(lbl_fn);
		vpgroup1.addComponent(txt_fn);
		vGroup.addGroup(vpgroup1);

		GroupLayout.ParallelGroup vpgroup2=g.createParallelGroup();
		vpgroup2.addComponent(lbl_sn);
		vpgroup2.addComponent(txt_sn);
		vGroup.addGroup(vpgroup2);

		GroupLayout.ParallelGroup vpgroup3=g.createParallelGroup();
		vpgroup3.addComponent(lbl_res);
		vpgroup3.addComponent(txt_res);
		vGroup.addGroup(vpgroup3);

		vGroup.addComponent(btn_add);

		g.setHorizontalGroup(hGroup);
		g.setVerticalGroup(vGroup);

		setVisible(true);
```

```
        }

        public static void main(String [] args)
        {
                new Demo();
        }
}
```