Diseño de formularios basados en Layouts

- *Layout Managers*

Layout Managers

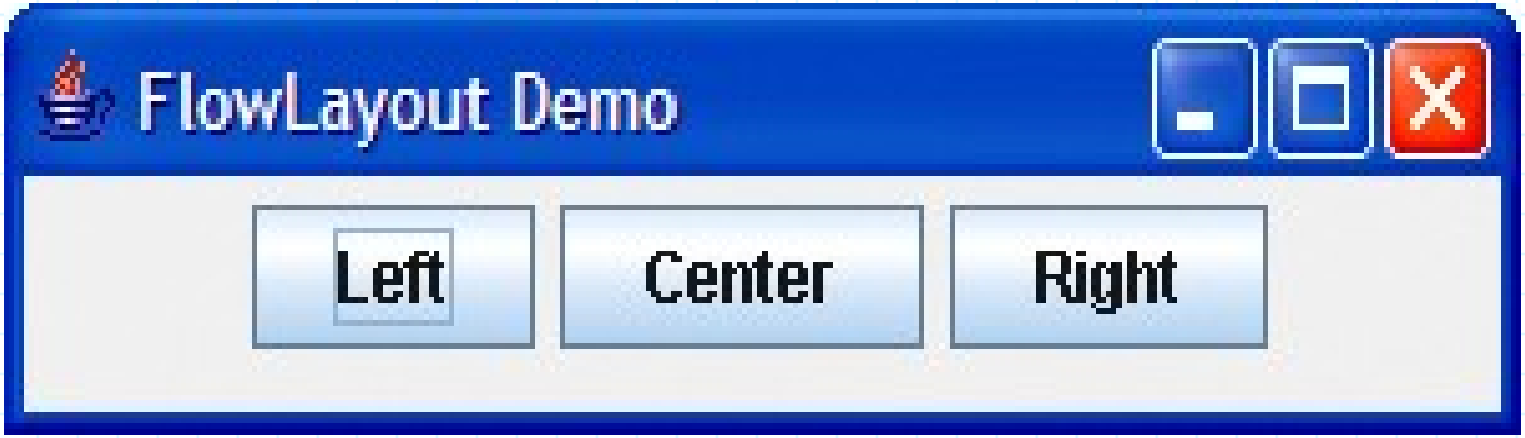
- Layout managers
  - Provisto para ordenar los componentes GUI en un contenedor.
  - Implementa la interface `LayoutManager`

Layout manager	Descripción
`FlowLayout`	Por defecto para `javax.swing.JPanel` Ubica los componentes secuencialmente (de izquierda a derecha) en el órden en que son agregados. Se puede especificar también el órden específico en que se desea incorporar el componente indicando en el método add la posición como segundo argumento.
`BorderLayout`	Por defecto para `JFrame`s (y otras ventanas). Arregla los componentes en cinco áreas: `NORTH`, `SOUTH`, `EAST`, `WEST` y `CENTER`.
`GridLayout`	Arregla los componentes en filas y columnas.

FlowLayout

- FlowLayout
  - Es el layout manager más simple
  - Los componentes son ubicados de izquierda a derecha a medida que se van agregando.
  - Los componentes pueden ser alineados a izquierda, derecha o centrados

FlowLayout Demo



```
1  // Fig. 11.39: FlowLayoutFrame.java
2  // Demonstrating FlowLayout alignments.
3  import java.awt.FlowLayout;
4  import java.awt.Container;
5  import java.awt.event.ActionListener;
6  import java.awt.event.ActionEvent;
7  import javax.swing.JFrame;
8  import javax.swing.JButton;
9
10 public class FlowLayoutFrame extends JFrame
11 {
12    private JButton leftJButton; // button to set alignment left
13    private JButton centerJButton; // button to set alignment center
14    private JButton rightJButton; // button to set alignment right
15    private FlowLayout layout; // layout object
16    private Container container; // container to set layout
17
18    // set up GUI and register button listeners
19    public FlowLayoutFrame()
20    {
21       super( "FlowLayout Demo" );
22
23       layout = new FlowLayout(); // create FlowLayout
24       container = getContentPane(); // get container to layout
25       setLayout( layout ); // set frame layout
26
```

Crear el `FlowLayout`

Establecer el layout de la aplicación

```
27      // set up leftJButton and register listener
28      leftJButton = new JButton( "Left" ); // create Left button
29      add( leftJButton ); // add Left button to frame
30      leftJButton.addActionListener(
31
32         new ActionListener() // anonymous inner cla
33         {
34            // process leftJButton event
35            public void actionPerformed( ActionEvent event )
36            {
37               layout.setAlignment( FlowLayout.LEFT );
38
39               // realign attached components
40               layout.layoutContainer( container );
41            } // end method actionPerformed
42         } // end anonymous inner class
43      ); // end call to addActionListener
44
45      // set up centerJButton and register listener
46      centerJButton = new JButton( "Center" ); // create Center button
47      add( centerJButton ); // add Center button to frame
48      centerJButton.addActionListener(
49
50         new ActionListener() // anonymous inner cla
51         {
52            // process centerJButton event
53            public void actionPerformed( ActionEvent event )
54            {
55               layout.setAlignment( FlowLayout.CENTER );
56
```

Agragar un `JButton`; `FlowLayout` se encargará de ubicarlo

Establecer la alineación a izquierda

Ajustar el layout

Agregar un `JButton`; `FlowLayout` se encargará de ubicarlo

Establecer la alineación centrada

Adjust layout

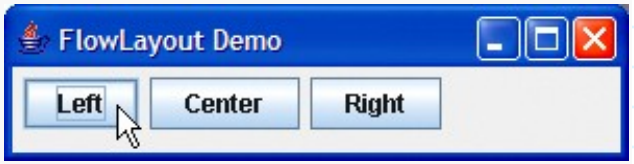
```
57                 // realign attached components
58                 layout.layoutContainer( container );
59              } // end method actionPerformed
60           } // end anonymous inner class
61        ); // end call to addActionListener
62
63        // set up rightJButton and register listener
64        rightJButton = new JButton( "Right" ); // create Right button
65        add( rightJButton ); // add Right button to frame
66        rightJButton.addActionListener(
67
68           new ActionListener() // anonymous inner cla
69           {
70              // process rightJButton event
71              public void actionPerformed( ActionEvent event )
72              {
73                 layout.setAlignment( FlowLayout.RIGHT );
74
75                 // realign attached components
76                 layout.layoutContainer( container );
77              } // end method actionPerformed
78           } // end anonymous inner class
79        ); // end call to addActionListener
80     } // end FlowLayoutFrame constructor
81  } // end class FlowLayoutFrame
```

Agregar un `JButton`; `FlowLayout` se encargará de ubicarlo

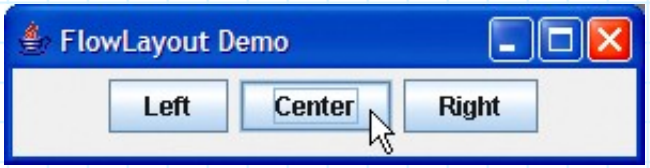
Establecer la alineación a Derecha

Ajustar el layout

```java
// Fig. 11.40: FlowLayoutDemo.java
// Testing FlowLayoutFrame.
import javax.swing.JFrame;

public class FlowLayoutDemo
{
   public static void main( String args[] )
   {
      FlowLayoutFrame flowLayoutFrame = new FlowLayoutFrame();
      flowLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      flowLayoutFrame.setSize( 300, 75 ); // set frame size
      flowLayoutFrame.setVisible( true ); // display frame
   } // end main
} // end class FlowLayoutDemo
```




FlowLayout Demo
Left
Center
Right

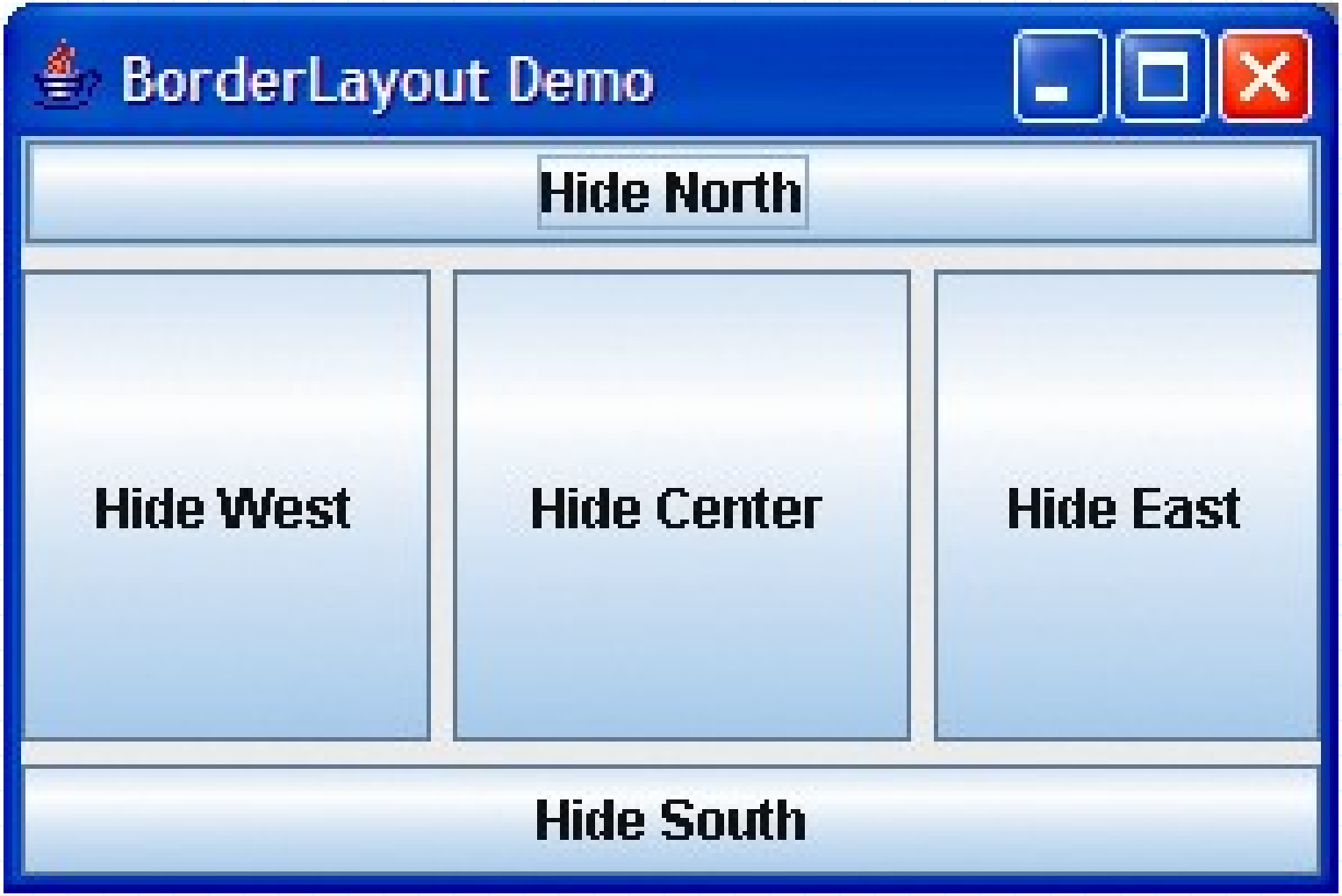

FlowLayout Demo
Left
Center
Right


FlowLayout D...
Left
Center
Right

BorderLayout

- BorderLayout
  - Ordena los componentes en cinco regiones: north, south, east, west y center
  - Implementa la interface LayoutManager2
  - Provee espaciado horizontal y vertical (gap)

BorderLayout Demo



```java
// Fig. 11.41: BorderLayoutFrame.java
// Demonstrating BorderLayout.
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JButton;

public class BorderLayoutFrame extends JFrame implements ActionListener
{
   private JButton buttons[]; // array of buttons to hide portions
   private final String names[] = { "Hide North", "Hide South",
      "Hide East", "Hide West", "Hide Center" };
   private BorderLayout layout; // borderlayout object

   // set up GUI and event handling
   public BorderLayoutFrame()
   {
      super( "BorderLayout Demo" );

      layout = new BorderLayout( 5, 5 ); // 5 pixel gaps
      setLayout( layout ); // set frame layout
      buttons = new JButton[ names.length ]; // set size of array

      // create JButtons and register listeners for them
      for ( int count = 0; count < names.length; count++ )
      {
         buttons[ count ] = new JButton( names[ count ] );
         buttons[ count ].addActionListener( this );
      } // end for
```

Decara una variable instancia de `BorderLayout`

Crea un `BorderLayout`

Establece el layout

Registrar los eventos

```
31
32        add( buttons[ 0 ], BorderLayout.NORTH ); // add button to north
33        add( buttons[ 1 ], BorderLayout.SOUTH ); // add button to south
34        add( buttons[ 2 ], BorderLayout.EAST ); // add button to east
35        add( buttons[ 3 ], BorderLayout.WEST ); // add button to west
36        add( buttons[ 4 ], BorderLayout.CENTER ); // add button to center
37     } // end BorderLayoutFrame constructor
38
39     // handle button events
40     public void actionPerformed( ActionEvent event )
41     {
42        // check event source and layout content pane correspondingly
43        for ( JButton button : buttons )
44        {
45           if ( event.getSource() == button )
46              button.setVisible( false ); // hide button clicked
47           else
48              button.setVisible( true ); // show other buttons
49        } // end for
50
51        layout.layoutContainer( getContentPane() ); // layout content pane
52     } // end method actionPerformed
53  } // end class BorderLayoutFrame
```

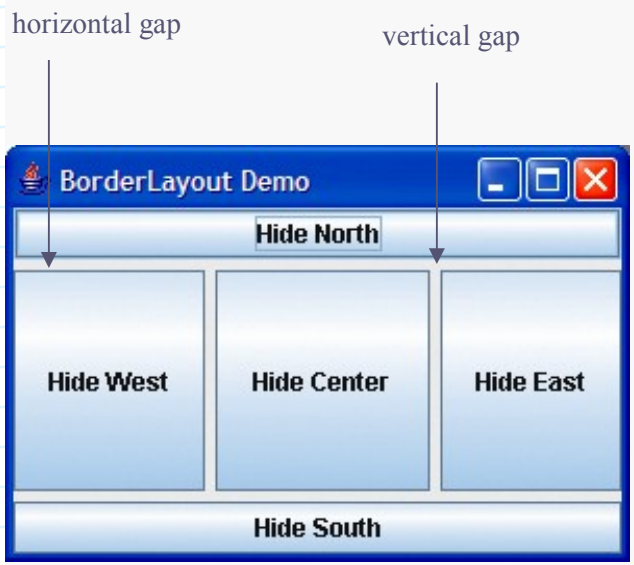
Agregar los botones usando las constantes del LM

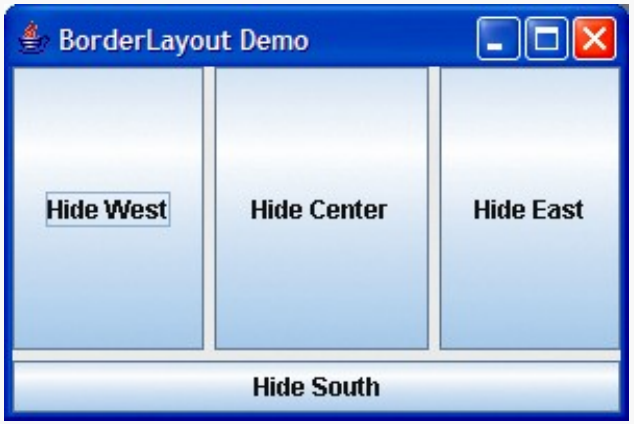
Hacer los botones invisibles

Hacer los botones visibles

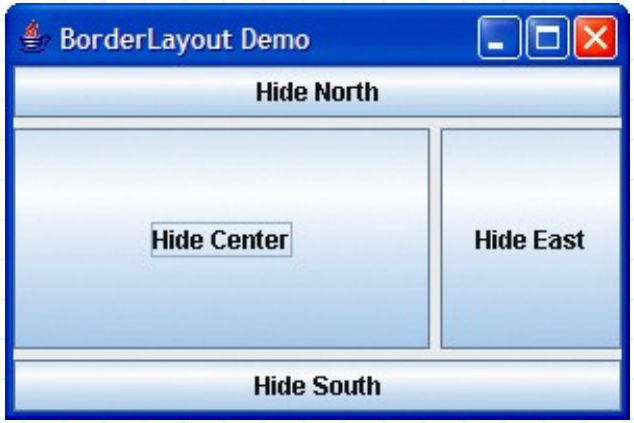
Actualizar el layout

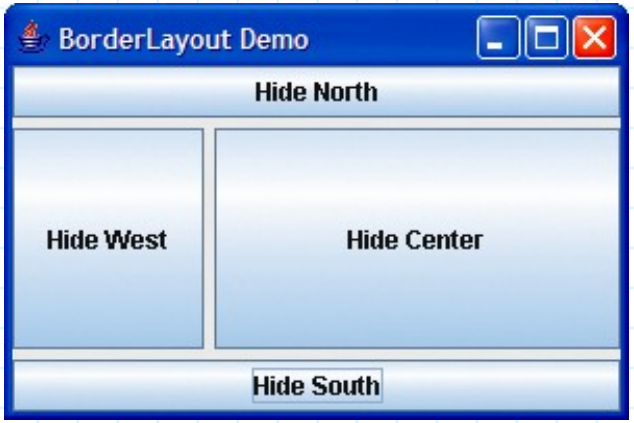
```java
// Fig. 11.42: BorderLayoutDemo.java
// Testing BorderLayoutFrame.
import javax.swing.JFrame;

public class BorderLayoutDemo
{
   public static void main( String args[] )
   {
      BorderLayoutFrame borderLayoutFrame = new BorderLayoutFrame();
      borderLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      borderLayoutFrame.setSize( 300, 200 ); // set frame size
      borderLayoutFrame.setVisible( true ); // display frame
   } // end main
} // end class BorderLayoutDemo
```

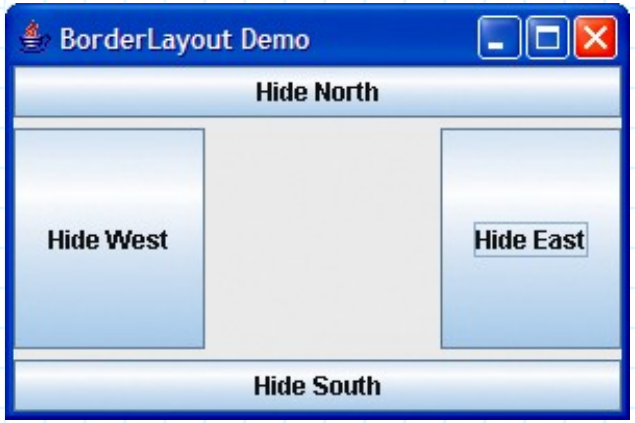





BorderLayout Demo
Hide North
Hide West
Hide Center
Hide East


BorderLayout Demo
Hide North
Hide Center
Hide East
Hide South

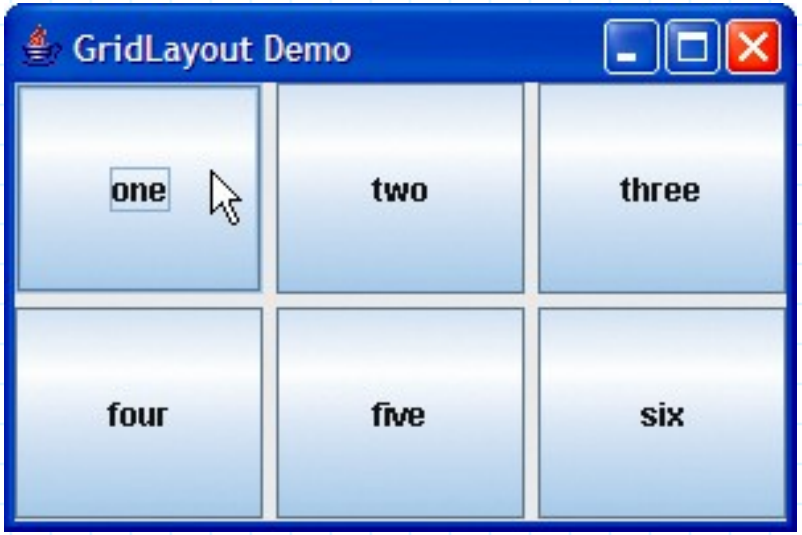

BorderLayout Demo
Hide North
Hide West
Hide Center
Hide South

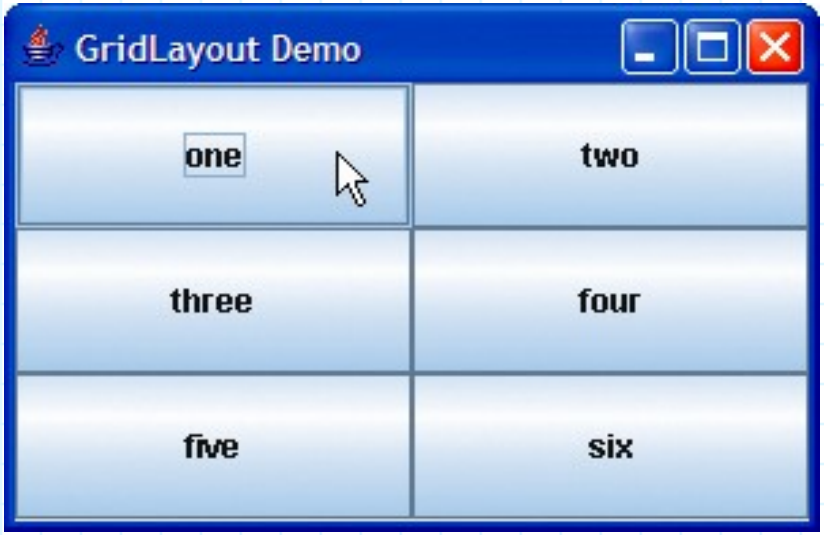

BorderLayout Demo
Hide North
Hide West
Hide East
Hide South

GridLayout

- GridLayout
  - Divide el contenedor en una grilla.
  - Todos los componentes tienen el mismo alto y ancho.

GridLayout Demo





```java
// Fig. 11.43: GridLayoutFrame.java
// Demonstrating GridLayout.
import java.awt.GridLayout;
import java.awt.Container;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JButton;

public class GridLayoutFrame extends JFrame implements ActionListener
{
   private JButton buttons[]; // array of buttons
   private final String names[] =
      { "one", "two", "three", "four", "five", "six"
   private boolean toggle = true; // toggle between t
   private Container container; // frame container
   private GridLayout gridLayout1; // first gridlayout
   private GridLayout gridLayout2; // second gridlayout

   // no-argument constructor
   public GridLayoutFrame()
   {
      super( "GridLayout Demo" );
      gridLayout1 = new GridLayout( 2, 3, 5, 5 ); // 2 by 3; gaps of 5
      gridLayout2 = new GridLayout( 3, 2 ); // 3 by 2; no gaps
      container = getContentPane(); // get content pane
      setLayout( gridLayout1 ); // set JFrame layout
      buttons = new JButton[ names.length ]; // creat

```

Declarar dos variables `GridLayout`

Crear los `GridLayout`

Establecer el layout

```
30        for ( int count = 0; count < names.length; count++ )
31        {
32           buttons[ count ] = new JButton( names[ count ] );
33           buttons[ count ].addActionListener( this ); // register listener
34           add( buttons[ count ] ); // add button to JFrame
35        } // end for
36     } // end GridLayoutFrame constructor
37
38     // handle button events by toggling between layouts
39     public void actionPerformed( ActionEvent event )
40     {
41        if ( toggle )
42           container.setLayout( gridLayout2 ); // set l
43        else
44           container.setLayout( gridLayout1 ); // set layout to first
45
46        toggle = !toggle; // set toggle to opposite value
47        container.validate(); // re-layout container
48     } // end method actionPerformed
49  } // end class GridLayoutFrame
```

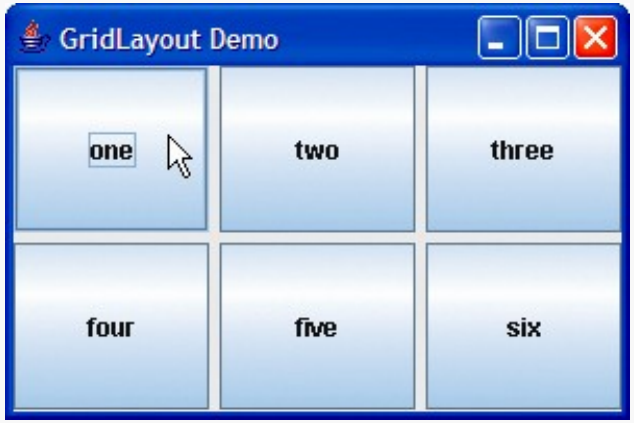
Agregar botones al `JFrame`

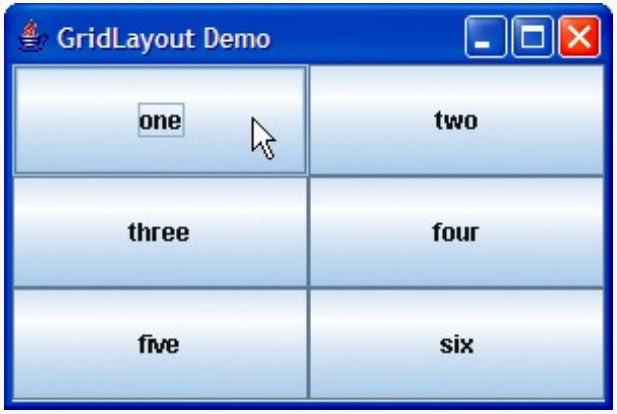
Usar el segundo layout

Usar el primer layout

Actualizar el layout

```java
// Fig. 11.44: GridLayoutDemo.java
// Testing GridLayoutFrame.
import javax.swing.JFrame;

public class GridLayoutDemo
{
   public static void main( String args[] )
   {
      GridLayoutFrame gridLayoutFrame = new GridLayoutFrame();
      gridLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      gridLayoutFrame.setSize( 300, 200 ); // set frame size
      gridLayoutFrame.setVisible( true ); // display frame
   } // end main
} // end class GridLayoutDemo
```





Usando Paneles para manejar layouts más complejos

- Los GUIs complicados normalmente requieren múltiples paneles para ordenar sus componentes adecuadamente

```java
// Fig. 11.45: PanelFrame.java
// Using a JPanel to help lay out components.
import java.awt.GridLayout;
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;

public class PanelFrame extends JFrame
{
   private JPanel buttonJPanel; // panel to hold buttons
   private JButton buttons[]; // array of buttons

   // no-argument constructor
   public PanelFrame()
   {
      super( "Panel Demo" );
      buttons = new JButton[ 5 ]; // create buttons array
      buttonJPanel = new JPanel(); // set up panel
      buttonJPanel.setLayout( new GridLayout( 1, buttons.length ) );

```

Declarar un `JPanel` para contener a los botones

Crear el `JPanel`

Establecer el layout

```
22        // create and add buttons
23        for ( int count = 0; count < buttons.length; count++ )
24        {
25           buttons[ count ] = new JButton( "Button " + ( count + 1 ) );
26           buttonJPanel.add( buttons[ count ] ); // add button to panel
27        } // end for
28
29        add( buttonJPanel, BorderLayout.SOUTH ); // add panel to JFrame
30     } // end PanelFrame constructor
31  } // end class PanelFrame
```

Agregar botones al panel

Agregar el panel a la aplicación

```java
// Fig. 11.46: PanelDemo.java
// Testing PanelFrame.
import javax.swing.JFrame;

public class PanelDemo extends JFrame
{
   public static void main( String args[] )
   {
      PanelFrame panelFrame = new PanelFrame();
      panelFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      panelFrame.setSize( 450, 200 ); // set frame size
      panelFrame.setVisible( true ); // display frame
   } // end main
} // end class PanelDemo
```