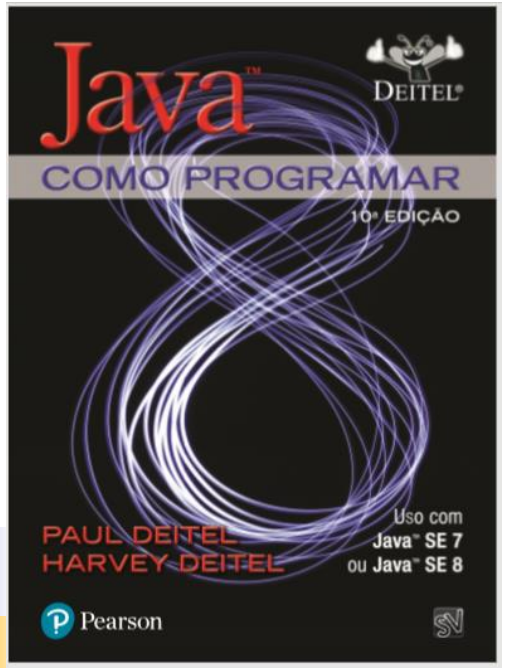


Java – Aula 6
Gerenciadores de Layout

Notas de Aula

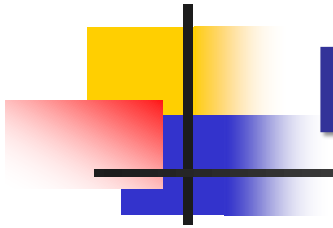
Prof. André Bernardi

andrebernardi@unifei.edu.br

outubro - 2021

Introdução a Gerenciadores de Layout

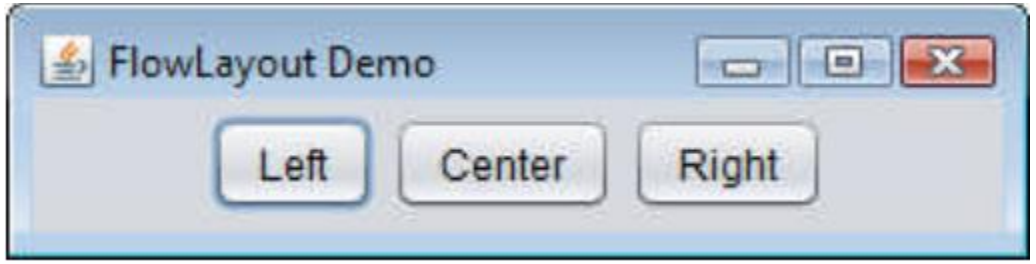
Gerenciador de layout	Descrição
`FlowLayout`	Padrão para `javax.swing.JPanel`. Coloca os componentes *sequencialmente, da esquerda para a direita*, na ordem em que foram adicionados. Também é possível especificar a ordem dos componentes utilizando o método `add` de `Container`, que aceita um `Component` e uma posição de índice do tipo inteiro como argumentos.
`BorderLayout`	Padrão para `JFrames` (e outras janelas). Organiza os componentes em cinco áreas: `NORTH`, `SOUTH`, `EAST`, `WEST` e `CENTER`.
`GridLayout`	Organiza os componentes nas linhas e colunas.



Layout

- **FlowLayout** – Padrão em painéis, onde um componente é adicionado ao lado do outro.
- **BorderLayout** – Padrão em Frames, onde os componentes ocupam regiões especificas no container.
- **GridLayout –** Divide o container em linhas e colunas.

FlowLayout




```
1 // Figura 12.39: FlowLayoutFrame.java
2 // FlowLayout permite que os componentes fluam ao longo de múltiplas linhas.
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 final JButton leftJButton;    // botão para configurar alinhamento à esquerda
13     private final JButton centerJButton;  // botão para configurar alinhamento centralizado
14     private final JButton rightJButton;   // botão para configurar alinhamento à direita
15     private final FlowLayout layout;      // objeto de layout
16     private final Container container;    // contêiner para configurar layout
17
```

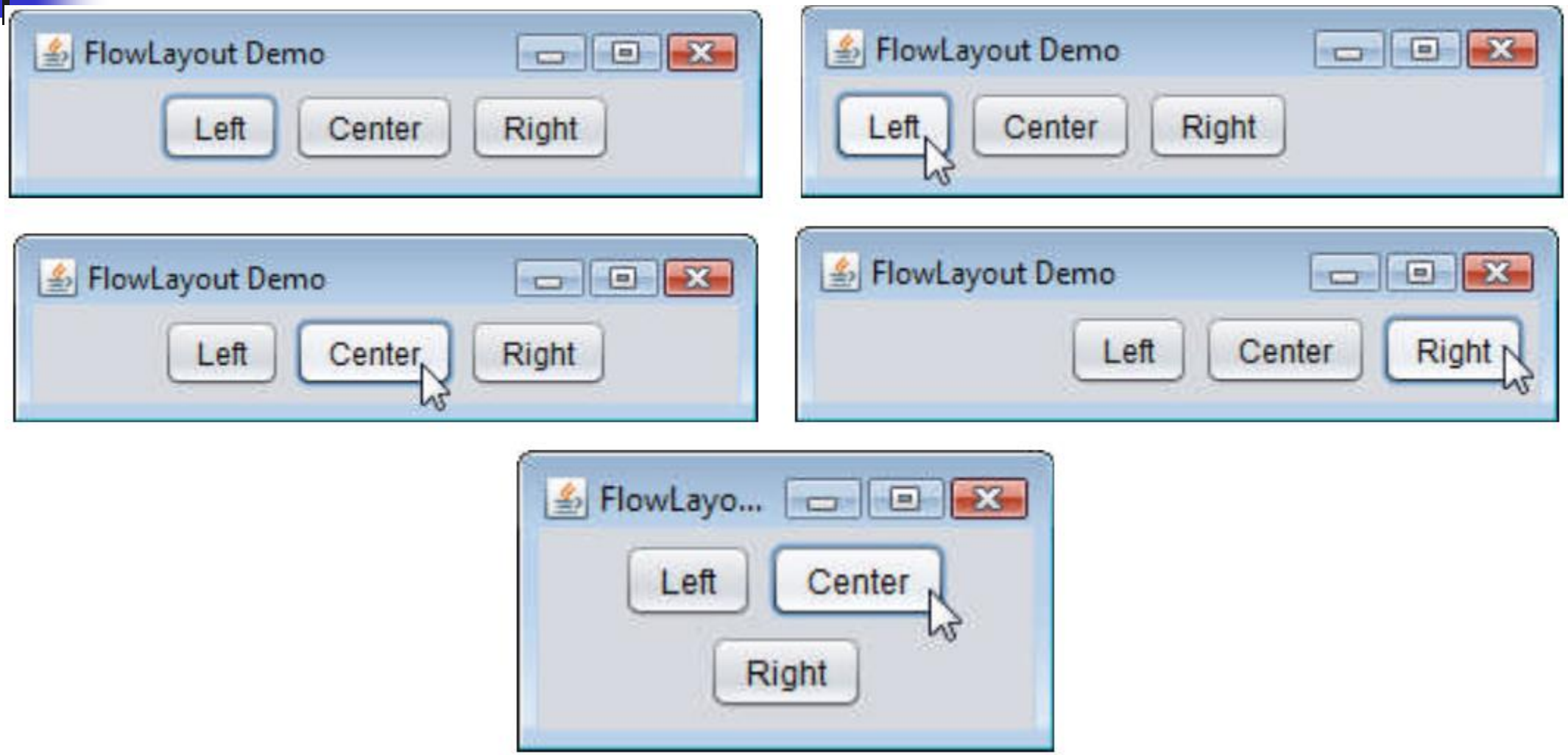
```
18      // configura GUI e registra listeners de botão
19      public FlowLayoutFrame()
20      {
21              super("FlowLayout Demo");      // chamada do construtor da classe base
22
23              layout = new FlowLayout();
24              container = getContentPane(); // obtém contêiner para layout
25              setLayout(layout);
26
27              // configura leftJButton e registra listener
28              leftJButton = new JButton("Left");
29              add(leftJButton); // adiciona o botão Left ao frame
30              leftJButton.addActionListener(
31                      new ActionListener() // classe interna anônima
32                      {       // processa o evento leftJButton
34                              @Override
35                              public void actionPerformed(ActionEvent event)
36                              {
37                                      layout.setAlignment(FlowLayout.LEFT);
38
39                                      // realinha os componentes anexados
40                                      layout.layoutContainer(container);
41                              }
42                      }
43              );
```

```
44
45          // configura centerJButton e registra o listener
46          centerJButton = new JButton("Center");
47          add(centerJButton); // adiciona botão Center ao frame
48          centerJButton.addActionListener(
49                  new ActionListener() // classe interna anônima
50                  {
51                          // processa evento centerJButton
52                          @Override
53                          public void actionPerformed(ActionEvent event)
54                          {
55                                  layout.setAlignment(FlowLayout.CENTER);
56
57                                  // realinha os componentes anexados
58                                  layout.layoutContainer(container);
59                          }
60                  }
61          );
62
63          // configura rightJButton e registra o listener
64          rightJButton = new JButton("Right");
65          add(rightJButton);                      // adiciona botão Right ao frame
66          rightJButton.addActionListener(
67                  new ActionListener()            // classe interna anônima
68                  {                               // processa evento rightJButton
```

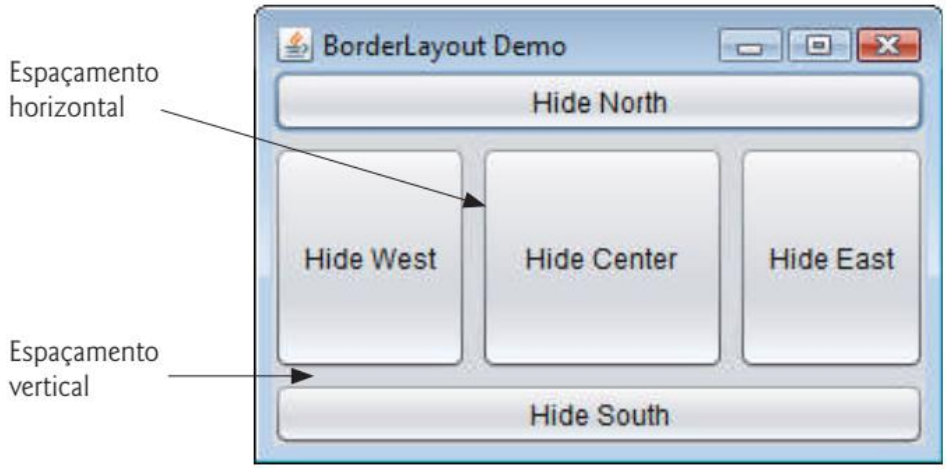
```
70                              @Override
71                              public void actionPerformed(ActionEvent event)
72                              {
73                                      layout.setAlignment(FlowLayout.RIGHT);
74
75                                      // realinha os componentes anexados
76                                      layout.layoutContainer(container);
77                      }       }
78              );
79      } // fim do construtor FlowLayoutFrame
80 } // fim da classe FlowLayoutFrame
```

```
1 // Figura 12.40: FlowLayoutDemo.java
2 // Testando FlowLayoutFrame.
3 import javax.swing.JFrame;
4 public class FlowLayoutDemo
5 {
6       public static void main(String[] args)
7       {
8               FlowLayoutFrame flowLayoutFrame = new FlowLayoutFrame();
9               flowLayoutFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10              flowLayoutFrame.setSize(300, 75);
11              flowLayoutFrame.setVisible(true);
12 }   } // fim da classe FlowLayoutDemo
```

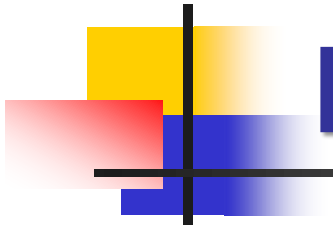
FlowLayout


FlowLayout Demo
Left
Center
Right
FlowLayout Demo
Left
Center
Right
FlowLayout Demo
Left
Center
Right
FlowLayout Demo
Left
Center
Right
FlowLayo...
Left
Center
Right

BorderLayout

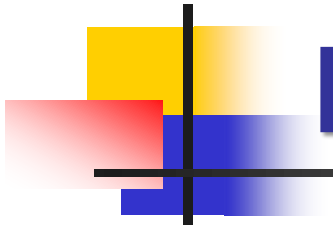


- O gerenciador de layout BorderLayout (o gerenciador de layout padrão de um JFrame) organiza componentes em cinco regiões: NORTH, SOUTH, EAST, WEST e CENTER.
- Um BorderLayout limita um Container a conter no máximo cinco componentes — um em cada região.



BorderLayout

- Se nenhuma região for especificada ao se adicionar um *Component* a um *BorderLayout*, o gerenciador de layout supõe que o *Component* deve ser adicionado à região *BorderLayout.CENTER*
- Quando mais de um componente for adicionado a uma região em um *BorderLayout*, somente o **último** componente adicionado a essa região será exibido. Não há nenhum erro que indica esse problema.

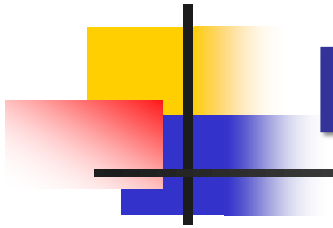


BorderLayout

```
1 // Figura 12.41: BorderLayoutFrame.java
2 // BorderLayout contendo cinco botões.
3 import java.awt.BorderLayout;
4 import java.awt.event.ActionListener;
5 import java.awt.event.ActionEvent;
6 import javax.swing.JFrame;
7 import javax.swing.JButton;
8
9 public class BorderLayoutFrame extends JFrame implements ActionListener
10 {
11    private final JButton[] buttons;      // array de botões para ocultar partes
12    private static final String[] names = {"Hide North", "Hide South",
13                                           "Hide East", "Hide West", "Hide Center"};
14    private final BorderLayout layout;
15
16    // configura GUI e tratamento de evento
17    public BorderLayoutFrame()
```

```
18      {
19              super("BorderLayout Demo");             // chamada do construtor da classe base
20
21              layout = new BorderLayout(5, 5);        // espaços de 5 pixels
22              setLayout(layout);
23              buttons = new JButton[names.length];
24
25              // cria JButtons e registra ouvintes para eles
26              for (int count = 0; count < names.length; count++)
27              {
28                      buttons[count] = new JButton(names[count]);
29                      buttons[count].addActionListener(this);
30              }
31
32              // adiciona os botões na janela
33              add(buttons[0], BorderLayout.NORTH);
34              add(buttons[1], BorderLayout.SOUTH);
35              add(buttons[2], BorderLayout.EAST);
36              add(buttons[3], BorderLayout.WEST);
37              add(buttons[4], BorderLayout.CENTER);
38      }
```

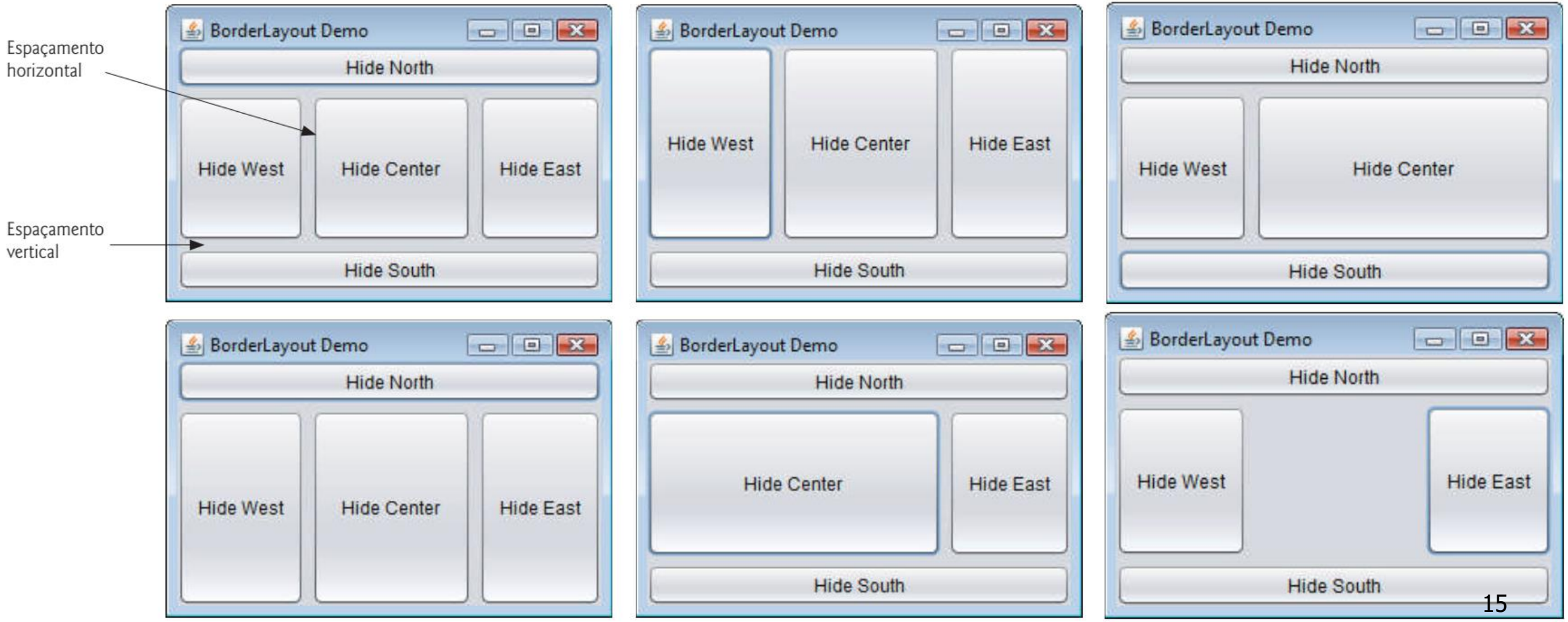
```
39      // trata os eventos de botão
40      @Override
41      public void actionPerformed(ActionEvent event)
42      {
43              // verifica a origem de evento e o painel de conteúdo de layout de acordo
44              for (JButton button : buttons)
45              {
46                      if (event.getSource() == button)
47                              button.setVisible(false); // oculta o botão que foi clicado
48                      else
49                              button.setVisible(true); // mostra outros botões
50              }
51
52              layout.layoutContainer(getContentPane()); // define o layout do painel de conteúdo
53      }
54  } // fim da classe BorderLayoutFrame
```



BorderLayout

```
1 // Figura 12.42: BorderLayoutDemo.java
2 // Testando BorderLayoutFrame.
3 import javax.swing.JFrame;
4
5 public class BorderLayoutDemo
6 {
7       public static void main(String[] args)
8       {
9             BorderLayoutFrame borderLayoutFrame = new BorderLayoutFrame();
10            borderLayoutFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11            borderLayoutFrame.setSize(300, 200);
12            borderLayoutFrame.setVisible(true);
13      }
14 } // fim da classe BorderLayoutDemo
```

BorderLayout



GridLayout

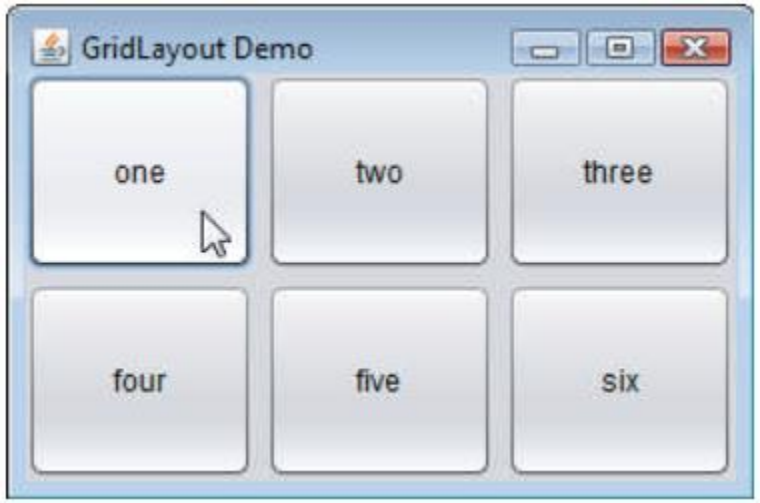
- O gerenciador de layout *GridLayout* divide o contêiner em uma grade de modo que os componentes podem ser colocados nas linhas e colunas.

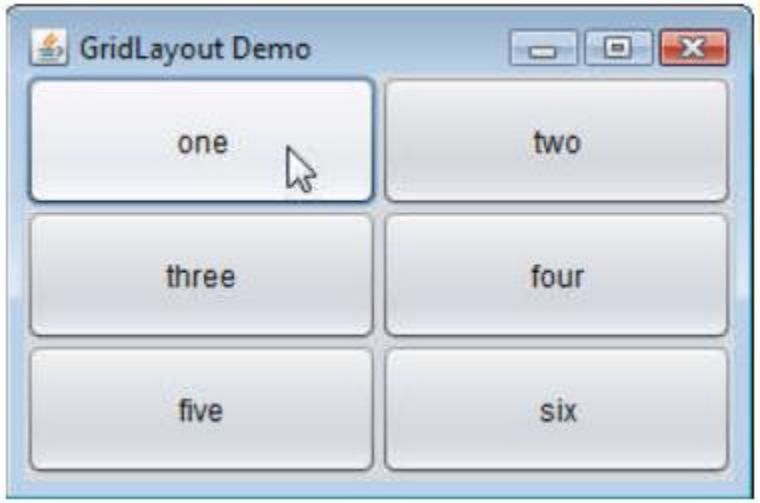
```
1 // Figura 12.43: GridLayoutFrame.java
2 // GridLayout contendo seis botões.
3 import java.awt.GridLayout;
4 import java.awt.Container;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ActionEvent;
7 import javax.swing.JFrame;
8 import javax.swing.JButton;
9
10 public class GridLayoutFrame extends JFrame implements ActionListener
11 {
12     private final JButton[] buttons;              // array de botões
13     private static final String[] names =
14                                   { "one", "two", "three", "four", "five", "six" };
15     private boolean toggle = true;                // alterna entre dois layouts
16     private final Container container;            // contêiner do frame
17     private final GridLayout gridLayout1;         // primeiro gridlayout
18     private final GridLayout gridLayout2;         // segundo gridlayout
19
20     // construtor sem argumento
21     public GridLayoutFrame()
22     {
23             super("GridLayout Demo");     // chamada do construtor da classe base
24             gridLayout1 = new GridLayout(2, 3, 5, 5);    // 2 por 3; lacunas de 5
25             gridLayout2 = new GridLayout(3, 2);          // 3 por 2; nenhuma lacuna
```

```
26              container = getContentPane();
27              setLayout(gridLayout1);
28              buttons = new JButton[names.length];
29
30              for (int count = 0; count < names.length; count++)
31              {
32                      buttons[count] = new JButton(names[count]);
33                      buttons[count].addActionListener(this);      // ouvinte registrado
34                      add(buttons[count]);                         // adiciona o botão ao JFrame
35              }
36      }
37
38      // trata eventos de botão alternando entre layouts
39      @Override
40      public void actionPerformed(ActionEvent event)
41      {
42              if (toggle)             // define layout com base nos botões de alternação
43                      container.setLayout(gridLayout2);
44              else
45                      container.setLayout(gridLayout1);
46
47              toggle = !toggle;
48              container.validate();         // refaz o layout do contêiner
49      }
50  } // fim da classe GridLayoutFrame
```

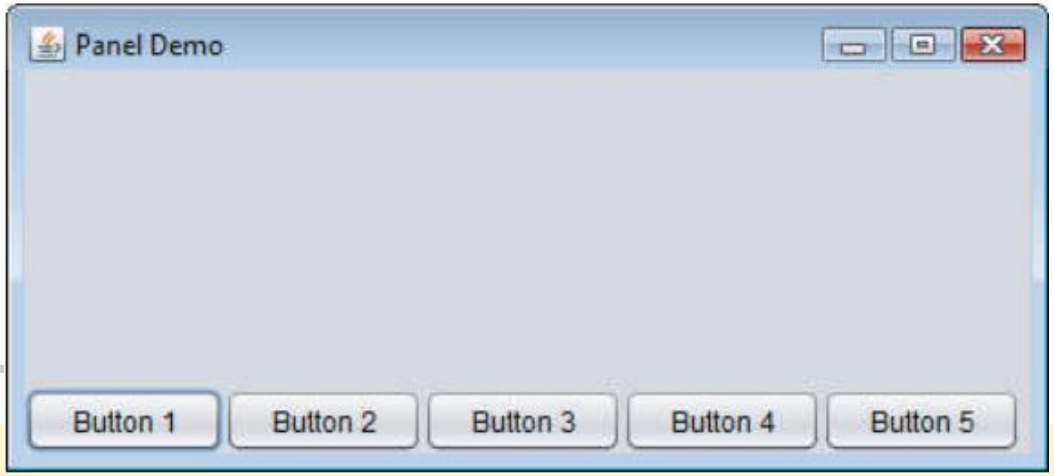
```java
// Figura 12.44: GridLayoutDemo.java
// Testando GridLayoutFrame.
import javax.swing.JFrame;

public class GridLayoutDemo
{
      public static void main(String[] args)
      {
            GridLayoutFrame gridLayoutFrame = new GridLayoutFrame();
            gridLayoutFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            gridLayoutFrame.setSize(300, 200);
            gridLayoutFrame.setVisible(true);
      }
} // fim da classe GridLayoutDemo
```



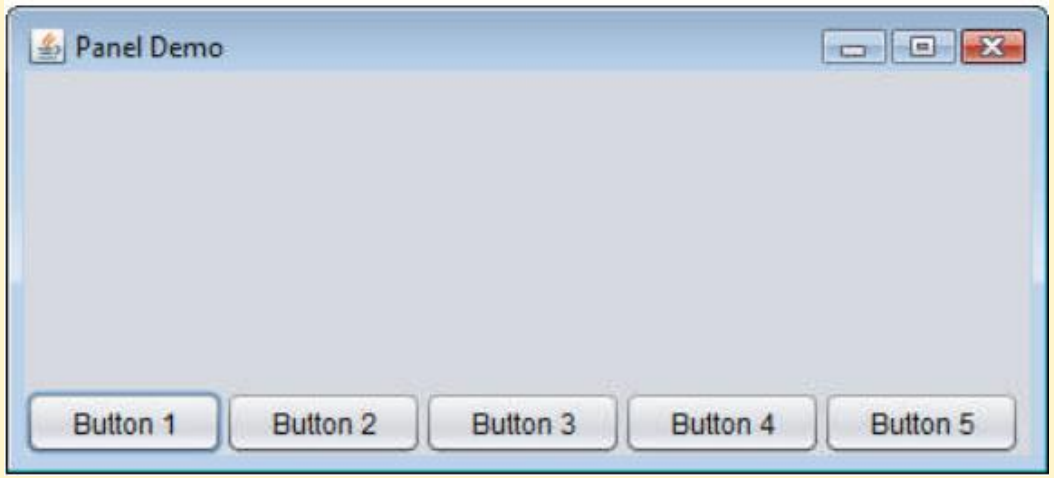


Usando Painéis




```
1 // Figura 12.45: PanelFrame.java
2 // Utilizando um JPanel para ajudar a fazer o layout dos componentes.
3 import java.awt.GridLayout;
4 import java.awt.BorderLayout;
5 import javax.swing.JFrame;
6 import javax.swing.JPanel;
7 import javax.swing.JButton;
8
9 public class PanelFrame extends JFrame
10 {
11     private final JPanel buttonJPanel;    // painel para armazenar botões
12     private final JButton[] buttons;
13
14     // construtor sem argumento
15     public PanelFrame()
16     {
17         super("Panel Demo");
```

```
18              buttons = new JButton[5];
19              buttonJPanel = new JPanel();
20              buttonJPanel.setLayout(new GridLayout(1, buttons.length));
22              // cria e adiciona botões
23              for (int count = 0; count < buttons.length; count++)
24              {
25                      buttons[count] = new JButton("Button " + (count + 1));
26                      buttonJPanel.add(buttons[count]);     // adiciona botão ao painel
27              }
29              add(buttonJPanel, BorderLayout.SOUTH);       // adiciona painel ao JFrame
30      }
31  } // fim da classe PanelFrame
```




```
1  // Figura 12.46: PanelDemo.java - Testando PanelFrame.
2  import javax.swing.JFrame;
3  public class PanelDemo extends JFrame
4  {
5       public static void main(String[] args)
6       {
7               PanelFrame panelFrame = new PanelFrame();
8               panelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9               panelFrame.setSize(450, 200);
10              panelFrame.setVisible(true);
11      }
```

Exercício

1) Utilizando Painéis e Layouts, montar a interface de uma calculadora conforme figura:

Exemplo Painter2


```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class Painter2 extends JFrame
6{
7     private final Color colorValues[] =  { Color.BLACK, Color.BLUE,
8           Color.RED, Color.GREEN, Color.YELLOW, Color.ORANGE, Color.PINK};
9     // array listing string colors
10    private final String colors[] = { "Black", "Blue",
11                                      "Red", "Green", "Yellow","Orange","Pink" };
12
13    private JRadioButton colorItems[];              // RadioButton para as cores
14    private ButtonGroup colorButtonGroup;           // manages colors items
15    private PaintPanel2 paintPanel,paintPanel2;     // cria o painel de pintura
```

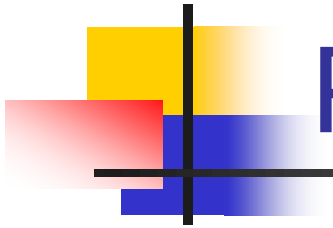
```
16    public static void main( String args[] )
17    {
18       // cria o JFrame
19       Painter2 application = new Painter2( );
20       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
21    } // fim de main
22
23    public Painter2 ()
24    {
25       super("Painter com cores");
26
27       // cria um painel para inserir dois componentes
28       JPanel painelCentral = new JPanel();
29       // muda o Layout para aceitar dois componentes
30       painelCentral.setLayout(new GridLayout(1,2));
31
32       // aloca os paineis de desenho
33       paintPanel = new PaintPanel2();
34       paintPanel2 = new PaintPanel2();
35
36      // adiciona eles no painel central
37      painelCentral.add(paintPanel);
38      painelCentral.add(paintPanel2);
```

```
39      //adiciona os paineis de desenho no centro
40      add(painelCentral, BorderLayout.CENTER);
41
42      // adiciona um label no sul do borderlayout
43      add(new JLabel("Arraste o mouse para desenhar."), BorderLayout.SOUTH);
44
45      // criando uma barra de ferramentas.
46      JPanel barraFerramentas = new JPanel();
47      // create radiobutton for colors
48      colorItems = new JRadioButton[ colors.length ];
49      // grupo logico para os botões
50      colorButtonGroup = new ButtonGroup(); // manages colors
51      // objeto para tratar o evento
52      ColorHandler colorHandler = new ColorHandler();
53
54      // create color radio buttons
55      for ( int count = 0 ; count < colors.length; count++ )
56      {
57         colorItems[ count ] =
58            new JRadioButton( colors[ count ] ); // create item
59         barraFerramentas.add( colorItems[ count ] ); // add item to color menu
60         colorButtonGroup.add( colorItems[ count ] ); // add to group
61         colorItems[ count ].addActionListener( colorHandler );
62      } // end for
```

```
63       colorItems[ 0 ].setSelected( true ); // select first Color item
64
65       // adiciona a barra de ferramentas no norte
66       add(barraFerramentas, BorderLayout.NORTH);
67       setSize(600, 300);
68       setVisible(true);
69    }
70
71     // inner class to handle action events from menu items
72     private class ColorHandler implements ActionListener
73     {  // process color and font selections
74        public void actionPerformed( ActionEvent event )
75        {  // process color selection
76           for ( int count = 0 ; count < colorItems.length; count++ ){
77              if ( colorItems[ count ].isSelected() ){
78                 paintPanel.setCorAtual(colorValues[ count ] );
79                 paintPanel2.setCorAtual(colorValues[ count ] );
80                 break ;
81              } // end if
82           } // end for
83           repaint(); // redraw application
84        } // end method actionPerformed
85     } // end class ColorHandler
86
87 } // end class Painter2
```

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 import java.util.*;
5
6 public class PaintPanel2 extends JPanel
7 {
8    // list Point references
9    private final ArrayList<Point> points = new ArrayList<>();
10    private final ArrayList<Color> cores = new ArrayList<>();
11    private Color corAtual = Color.BLACK;
12    // set up GUI and register mouse event handler
13    public PaintPanel2()
14    {
15       // handle frame mouse motion event
16       addMouseMotionListener( new MouseMotionAdapter() // anonymous inner class
17          {  // store drag coordinates and repaint
18             public void mouseDragged(MouseEvent event)
19             {
20                points.add(event.getPoint());
21                cores.add(corAtual);
22                repaint();
23             }
24          } );
25    } // end constructor
```

```
25    public void setCorAtual(Color c)
26    {
27       if( c != null )
28            corAtual = c;
29       else
30            corAtual = Color.BLACK;
31    }
32
33    // draw ovals in a 4-by-4 bounding box at specified locations on window
34    public void paintComponent(Graphics g)
35    {
36       super.paintComponent(g);              // clears drawing area
37
38      // draw all
39      int n = cores.size();
40      for (int i = 0; i < n; i++ )
41      {
42          Point point = points.get(i);
43          g.setColor( cores.get(i) );
44          g.fillOval(point.x, point.y, 4, 4);
45      }
46    }
47
48} // end class PaintPanel2
```



Referências

- Java How to Program 3 a 10 ed.

Paul Deitel e Harvey Deitel