



## Gerenciadores de Layout

---

Mário Antonio Meireles Teixeira  
DEINF – UFMA

Baseado em material original de  
João Carlos Pinheiro – CEFET/MA

1



## Objetivos

---

- Apresentar os gerenciadores de layout:
  - ◆ **FlowLayout**
  - ◆ **BorderLayout**
  - ◆ **GridLayout**
  - ◆ **BoxLayout**
  - ◆ **GridBagLayout**

2



## Gerenciadores de Layouts

- São fornecidos para **organizar** componentes GUI em um container para propósitos de apresentação
  - ◆ **processam** a maioria dos detalhes, fornecendo uma maneira **automática** de posicionar os componentes gráficos nos containers
- Todos os gerenciadores de Layout implementam a interface **LayoutManager** ou a subinterface **LayoutManager2**
- Cada container como um Painel ou Frame (molduras) possui um **gerenciador de layout** padrão associado

3



## Gerenciadores de Layouts

- É possível **desativar** o Gerenciador de Layout caso o programador queira definir o tamanho ou posição dos componentes
- Para **desligar** layouts:
  - ◆ `recipiente.setLayout(null);`
- Porém torna-se necessário **definir posição e tamanho** de cada componente:
  - ◆ `componente.setBounds(x, y, larg, alt);`

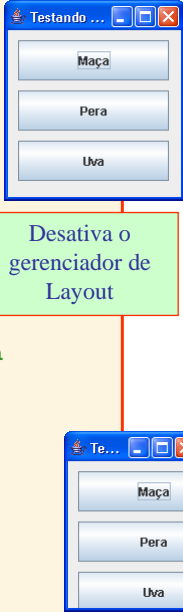
4

**Exemplo**

```

1. public class NullLayout extends JFrame {
2.     private JButton b1, b2, b3;
3.     NullLayout(String nome) {
4.         super(nome);
5.         Container ct = this.getContentPane();
6.         ct.setLayout(null);
7.         b1 = new JButton("Maça");
8.         b2 = new JButton("Pera");
9.         b3 = new JButton("Uva");
10.        // parametros: x, y, largura e altura
11.        b1.setBounds(10,10,150,40);
12.        b2.setBounds(10,60,150,40);
13.        b3.setBounds(10,110,150,40);
14.        ct.add(b1); ct.add(b2); ct.add(b3);
15.        this.setSize(180, 200);
16.        this.setVisible(true);
17.    } }

```



Desativa o gerenciador de Layout



## Tamanho e Posição dos Componentes

- Um container **mantém** uma referência a uma instância particular do Gerenciador de Layout
  - ◆ Sempre quando for necessário **posicionar** o componente será invocado o Gerenciador de Layout
  - ◆ Ou seja, se o usuário **tentar definir** o tamanho ou posição dos componentes (`setSize()` ou `setLocation()`) o Gerenciador de Layout irá **anular** a decisão



## Tamanho e Posição dos Componentes

```
public class TamanhoComp extends JFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        JPanel panel = new JPanel();
        for (int i=0; i<2; i++) {
            JButton btn = new JButton("Não adianta chamar setSize");
            btn.setSize(300, 300);
            panel.add(btn);
        }
        frame.getContentPane().add(panel);
        frame.setSize(450, 70);
        frame.setVisible(true);
    }
}
```

Esta linha, não tem efeito, pois o botão é adicionado ao painel, que chama o seu gerenciador de layout para decidir a sua posição e quão grande o componente deve ficar



7



## Política de Layout

- Cada componente Java tem um tamanho preferencial:

- ◆ Normalmente é o menor tamanho necessário para apresentar o componente de maneira visualmente significativa
  - ◆ Exemplo de um botão é o tamanho de sua etiqueta de texto

- O Gerenciador de Layout equilibra duas considerações: a **política de layout** e o **tamanho preferencial** de cada componente. A **prioridade** é para a política de layout

8



## Etapas para se construir uma interface

- Cada Painel é montado em **quatro** etapas:
  1. **Construir (instanciar) o painel**
  2. **Dar ao painel o gerenciador de layout**
    - ◆ Quando um container é instanciado (etapa 1) ele recebe um gerenciador de layout padrão. Esta etapa pode ser pulada se o gerenciador a **ser utilizado** for o padrão
  3. **Preencher o painel**
    - ◆ Envolve montar componentes e acrescentá-los ao painel
  4. **Acrescentar o painel ao seu próprio container**
- **OBS:** *Se algum dos componentes for ele próprio um painel, as etapas 1– 4 precisam ser executadas sucessivamente*

9



## Gerenciadores de Layouts

- Existem vários gerenciadores de layouts disponível com a linguagem Java
  - ◆ **FlowLayout, BorderLayout, GridLayout**
  - ◆ **BoxLayout, GridBagLayout**
- Principais métodos da classe `java.awt.Container`
  - ◆ **`setLayout (LayoutManager m)`**
  - ◆ **`void add(Component c)`** – adiciona um componente ao container sob o controle do gerenciador de layout utilizado

10



## FlowLayout

---

- É o gerenciador de layout mais simples
- Dispõe os objetos **seqüencialmente** da esquerda para a direita na ordem em que foram adicionados
  - ◆ `container.setLayout(new FlowLayout());`
- Permite aos componentes serem **alinhados** à esquerda, centralizados (padrão) ou à direita
  - ◆ `objetoLayout.setAlignment(FlowLayout.RIGHT);`

11



## FlowLayout

---

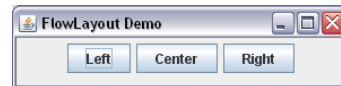
- Todos os componentes possuem um método chamado **getPreferredSize()**, usado pelos gerenciadores de layout para **perguntar** o tamanho de cada componente
- É o padrão para
  - ◆ `java.awt.Applet`,
  - ◆ `java.awt.Panel` e
  - ◆ `javax.swing.JPanel`

12



## FlowLayout - Principais construtores

- `FlowLayout()`
  - ◆ Centraliza os componentes deixando uma distância entre eles (**gap**) de 5 pixels.
- `FlowLayout(int align)`
  - ◆ Constrói um layout com alinhamento estabelecido (`FlowLayout.CENTER`, `FlowLayout.LEFT`, `FlowLayout.RIGHT`)
- `FlowLayout(int align, int hgap, int vgap)`
  - ◆ Define o alinhamento e a distância horizontal e vertical, respectivamente



Confira: `FlowLayoutDemo.java`

13



## BorderLayout

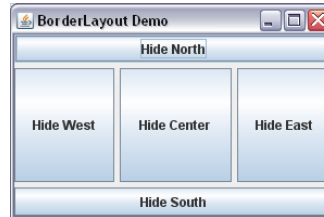
- É o gerenciador padrão para **JFrames**, **JApplets** e **Caixas de Diálogo**
- A classe **BorderLayout** herda diretamente da classe **Object** e implementa a interface **LayoutManager2** (uma subinterface de **LayoutManager**)
- Organiza os componentes em cinco áreas
  - ◆ Norte, Sul, Leste, Oeste e Centro

14



## BorderLayout

- Os componentes devem ser adicionadas a **regiões nomeadas** no gerenciador de layout, caso contrário não ficarão visíveis: NORTH, SOUTH, EAST, WEST, CENTER
- Norte e Sul têm prioridade sobre Leste e Oeste, que têm prioridade sobre Centro
- Se for adicionado um componente ao Centro, este se expande até ocupar todo o espaço restante na janela
- O componente colocado em uma região pode ser um container ao qual serão anexados outros componentes (p.ex., JPanel)



Confira: `BorderLayoutDemo.java`

15



## GridLayout

- Fornece **flexibilidade** para colocação de componentes
- **Divide** o container em uma **grade** de modo que os componentes podem ser colocados em linhas e colunas
- A classe `GridLayout` herda diretamente da classe `Object` e implementa a interface `LayoutManager`

16





## GridLayout

- Com o gerenciador `GridLayout`, a **posição relativa** dos componentes **não é alterada** quando a área é redimensionada
- Os componentes são **adicionados** iniciando na célula na parte **superior esquerda** da grade e prosseguindo da esquerda para a direita até a linha estar cheia. Então, o processo continua na próxima linha

Confira:  
`GridLayoutDemo.java`



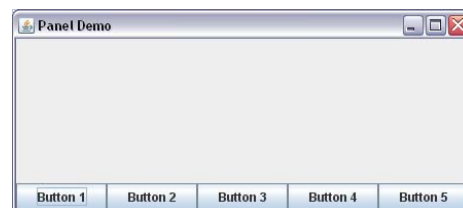
17



## Gerenciando layouts complexos com Painéis

- Interfaces complexas exigem que cada componente seja colocado numa **localização exata**
- Para tanto, geralmente usam-se múltiplos painéis, cada um com seu layout específico
- Container -> `JComponent` -> `JPanel`

Confira:  
`PanelDemo.java`



18



## Classe "container" Box

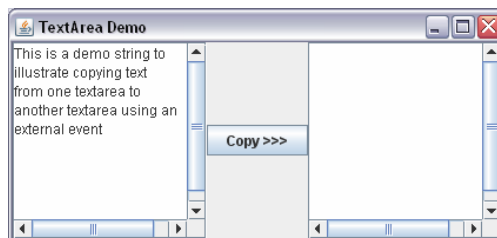
- Permite colocar uma única linha ou coluna de componentes com mais flexibilidade do que o `GridLayout`
- Existe um Container - a classe `Box` – cujo gerenciador de layout padrão é `BoxLayout` :
  - ◆ Fornece métodos estáticos para criar um `BoxLayout` horizontal ou vertical:
    - ◆ `Box b = Box.createHorizontalBox( )`
      - São organizados da esquerda para direita
    - ◆ `Box b = Box.createVerticalBox( )`
      - São organizados de cima para baixo

19



## Gerenciador BoxLayout e a classe "container" Box

- Depois, adicionam-se os componentes da maneira usual:
  - ◆ `b.add(btnOk);`
  - ◆ `b.add(btnSair);`



Confira: `TextAreaDemo.java`

20



## Referências

---

- Deitel & Deitel. Java como Programar, 6ª ed. Caps. 11 e 22;
- Materiais do site da SUN (<http://java.sun.com>)