MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS
Curso de Sistemas de Informação
Prof. Marcio Alves de Macedo
Aluno(a): Diego Fernando de Sousa Lima
Email: diegofernando5672@gmail.com

ATIVIDADE DE PROGRAMAÇÃO ORIENTADA A OBJETOS

TIPOS DE LAYOUTS EM JAVA

PICOS
2016

1. Gerenciadores de *layouts* em Java

Os gerenciadores de ***layouts*** em Java são responsáveis por dispor os componentes gráficos na tela e para isto existem algumas maneiras que são responsáveis pela organização dos elementos.

Neste mecanismo é onde fazemos as configurações dos componentes de um *frame* sendo eles do tipo AWT ou *Swing*. Configurações estas que são varias como: posicionamento, definição de tamanho, fonte, cor, entre vários outros parâmetros.

Ao escrever um aplicação usando interface gráfica, devemos definir qual ***Layout Manager*** devemos utilizar. Após instanciarmos um *frame* usamos uma série de métodos para definir as configurações desta janela e um destes é o `setLayout` que fica responsável justamente por qual *Layout Manager* iremos usar. Além deste método, temos outros da classe `Container` e detre os principais para este momento, temos: `setSize` ou `setBounds` para definir o tamanho do *frame*, `setLocation` para definir a localização onde vai ficar na tela do computador ao executarmos a aplicação, `setResizable` para definir se o *frame* pode ou não ser redimensionado pelo usuário final e o `setVisible` para tornarmos este *frame* visível na tela.

```
setLayout(new FlowLayout());
setTitle("Exemplo Flow Layout");
setDefaultCloseOperation(EXIT_ON_CLOSE);
setVisible(true);
setResizable(false);
setSize(350, 200);
```

(Exemplo de configurações do *layout*)

2. Tipos de *layouts*

2.1. *Layouts* gerenciáveis

Além de visual, umas das vantagens dos layouts gerenciáveis é a praticidade de configurar os componentes dentro do *frame*. Listaremos neste artigo três tipos de layouts gerenciáveis: **`FlowLayout, BorderLayout, GridLayout`**.

2.1.1. FlowLayout

Neste *Layout Manager* os elementos são colocados em ordem de adição e posicionados da esquerda para a direita. Quando os elementos alcançam a borda direita são lançados uma linha baixo dos primeiros componente assim redimensionando os elementos por igual e estando centralizado por padrão.

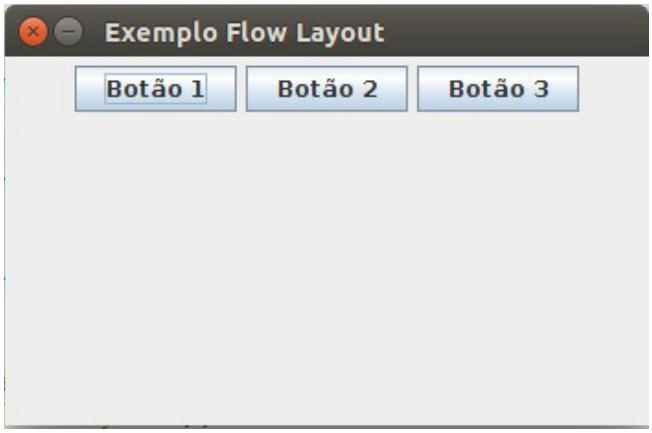
Este tipo também permite que os componentes sejam alinhados à esquerda, centralizados e alinhados à direita.

Para usarmos o ***FlowLayout*** na aplicação basta alterar o método de definição para new `FlowLayout()` e fica `setLayout(new FlowLayout())`.
Este é um exemplo de ***FlowLayout*** com três componentes do mesmo tipo (`JButton`):

```
public fl_layout(){

      botao1 = new JButton("Botão 1");
      add(botao1);
```

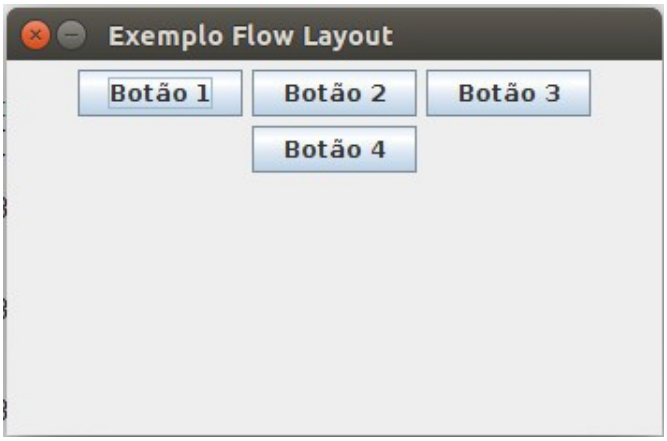
```
        botao2 = new JButton("Botão 2");
        add(botao2);

        botao3 = new JButton("Botão 3");
        add(botao3);

        setLayout(new FlowLayout());
        setTitle("Exemplo Flow Layout");
```

(Exemplo de *FlowLayout*)



(Interface *FlowLayout*)

Se colocarmos mais um botão:



(Interface com o 4º botão na linha de baixo)

Note que os itens estão centralizados. Outro método também útil do ***FlowLayout*** é `setAlignment` que é responsável pelo alinhamento dos elementos dentro do componente ***FlowLayout***. Este parâmetro também pode ser alterado diretamente no método *setLayout*: `setLayout(new FlowLayout(FlowLayout.(tipo_de_alinhamento)))`. Os tipos mais comuns de alinhamento além de `CENTER` (padrão), são o `LEFT` e o `RIGHT`.

```
setLayout(new FlowLayout(FlowLayout.(FlowLayout.LEFT)))
```

(Exemplo de alinhamento em FlowLayout)

2.1.2. BorderLayout

A principal função do ***BorderLayout*** é dividir os elementos do componente em cinco regiões: **NORTH**, **SOUTH**, **EAST**, **WEST** e **CENTER**.

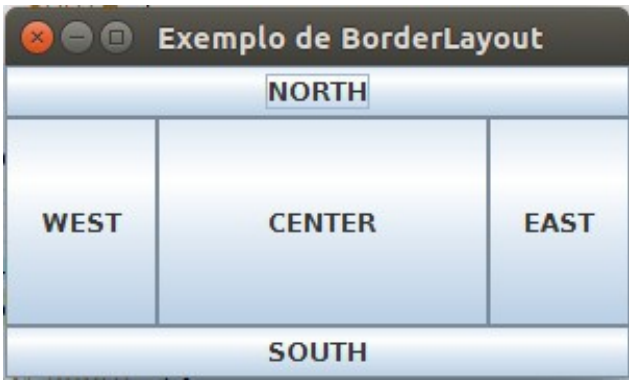
Diferentemente do ***FlowLayout***, o ***BorderLayout*** não traz padrões para os tamanhos dos componentes e isso quer dizer que os componentes são esticados de forma que fiquem justos aos outros. Quanto aos tamanhos dos botões é pratico mudar o tamanho para que o gerenciador de layout não tome de conta, basta configurar o valor `setSize()` dos mesmos.

Quanto a localização, o ***BorderLayout*** não foi feito para isso, porém, veremos isso neste artigo em "***Layout 'null'***".

Para configurarmos um componente para ***BorderLayout*** seguimos o mesmo modelo do ***FlowLayout*** com uma única diferença: neste modelo podemos configurar os espaços em *pixels* na declaração. Fica assim: `setLayout(new BorderLayout(x,y))`, em que 'x' configura os espaços laterais e o 'y' configura os lados de cima e de baixo.

```
public bd_layout(){

    setLayout(new BorderLayout());

    JButton bt1 = new JButton("CENTER");
    JButton bt2 = new JButton("NORTH");
    JButton bt3 = new JButton("SOUTH");
    JButton bt4 = new JButton("EAST");
    JButton bt5 = new JButton("WEST");

    add(BorderLayout.CENTER, bt1);
    add(BorderLayout.NORTH, bt2);
    add(BorderLayout.SOUTH, bt3);
    add(BorderLayout.EAST, bt4);
    add(BorderLayout.WEST, bt5);
```

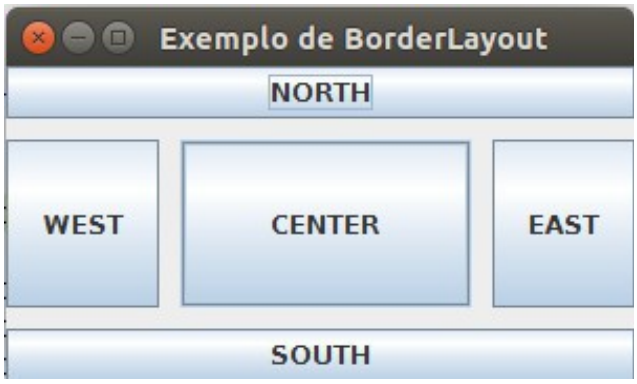
(*BorderLayout* sem espaçamento)



(*BorderLayout* sem espaçamento)

Para configurar o espaçamento:

```
setLayout(new BorderLayout(10,10));
```



(*BorderLayout* com espaçamento 10,10)

2.1.3. GridLayout

O ***GridLayout*** por sua vez divide o *container* em uma grade de modo que os componentes podem ser colocados em linhas e colunas.

Neste tipo os componentes devem ter o mesmo tamanho e os elementos são inseridos da esquerda para a direita. Este layout manager trabalha com células.

O ***GridLayout*** te dá a possibilidade de configurar a quantidade de linhas e colunas para o uso já na declaração. Para configurarmos um ***GridLayout*** fazemos igual aos anteriores: `setLayout(new GridLayout(x,y))` onde x e y são quantidade de linhas e colunas respectivamente. Lembrando que não é obrigatório definir o parâmetro de linhas e colunas.

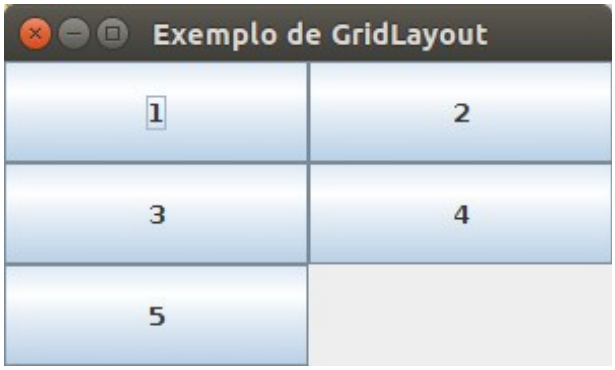
No primeiro exemplo demonstraremos como fica sem esses dois parâmetros.

```
public gd_layout(){

    setLayout(new GridLayout());

    JButton bt1 = new JButton("1");
    JButton bt2 = new JButton("2");
    JButton bt3 = new JButton("3");
    JButton bt4 = new JButton("4");
    JButton bt5 = new JButton("5");
    JButton bt6 = new JButton("6");

    add(bt1);
    add(bt2);
    add(bt3);
    add(bt4);
    add(bt5);
```

(*GridLayout* com linhas e colunas padrões)

Para configurar as linhas e colunas usamos:

```
setLayout(new GridLayout(3,2));
```



(*GridLayout* com linhas e colunas 3,2)

2.2. Layout 'null'

O ***layout null*** ou simplesmente 'nulo' se retrai de todos os aparatos descritos anteriormente e meio que anda na contramão da organização fácil dos elementos no componente. Este traz mobilidade e liberdade na hora de criação de interfaces gráficas até porque podemos colocar os elementos onde literalmente quisermos.

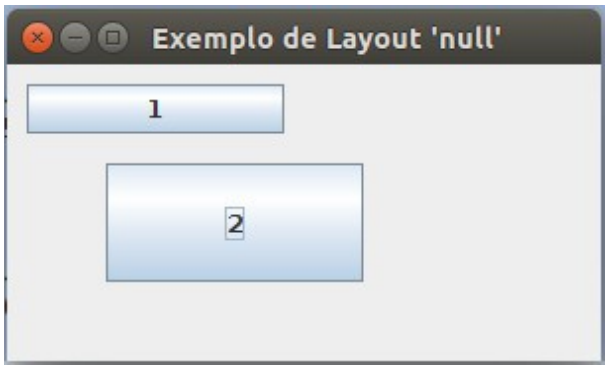
Há quem diga que os ***layouts null*** não são boas práticas de programação e há quem diga que não há nada a ver. Fica critério e bom senso do utilizador. O fato é que nenhum aparato será uma boa prática se não soubermos usar com total clareza e organização.

O ***layout null*** trás liberdade para disposição dos elementos, porém, nos damos de cara com, provavelmente, mais linhas de códigos. Isso se dá pelo fato que cada elemento dentro do componente deve ter uma configuração específica.

```
public null_layout(){

      setLayout(null);

      JButton bt1 = new JButton("1");
      bt1.setSize(130, 25);
      bt1.setLocation(10, 10);
      add(bt1);

      JButton bt2 = new JButton("2");
      bt2.setSize(130, 60);
      bt2.setLocation(50, 50);
      add(bt2);
```

(Exemplo de *GridLayout*)

Neste exemplo acima são colocados parâmetros totalmente sem qualquer senso de organização o resultado veremos:



(Exemplo de *layout null* desorganizado)

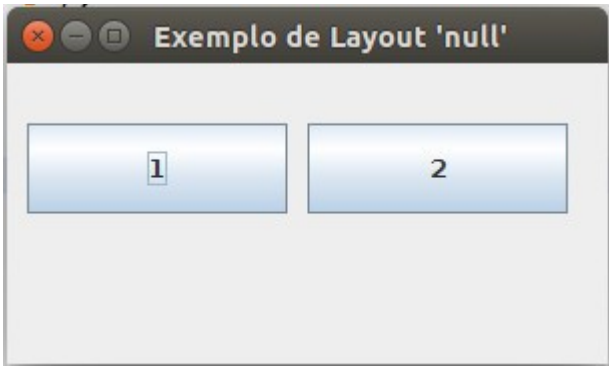
Se colocarmos o `setSize` do mesmo tamanho e configuramos o `setLocation` igual a:

```
bt1.setLocation(10, 30);
```

e

```
bt2.setLocation(150, 30);
```

Teremos este resultado:



(Exemplo de *layout null* organizado)

3. Conclusão

Portanto, os *Layout Managers* auxiliam bastante na organização dos componentes de um modo geral. Como saberemos qual utilizar? Isso vai depender da aplicação. Casos como calculadoras ou aplicações que exigem uma certa quantidade de botões em forma de célula, será bem mais prático trabalhar com ***GridLayout***. Percebe-se que o ***BorderLayout*** tem uma aparência mais voltada a um mecanismo de controle de algo. E o ***FlowLayout*** talvez seja o mais comum dentre os *layouts* gerenciáveis.

Apesar da suspeita de não ser uma boa prática de programação, o layout mais apropriado para organizar os elementos de forma mais precisa é o ***layout null*** pelos mesmos aspectos já relatados e por nos dar totais condições de configurarmos um *layout* à maneira que quisermos.

4. Referências

DevMedia. Conhecendo gerenciadores de Layout GUI em Java. Disponível em: <http://www.devmedia.com.br/conhecendo-gerenciadores-de-layout-gui-do-java/>. Acesso em: 12 de maio de 2016.

Portal Educação. Gerenciadores de Layouts em Java. Disponível em: <http://www.portaleducacao.com.br/informatica/artigos/4892/gerenciadores-de-layout-em-java>. Acesso em: 12 de maio de 2016.

Caelum. Mais swing: layout managers, mais componentes e detalhes. Disponível em: <http://www.caelum.com.br/apostila-java-testes-xml-design-patterns/mais-swing-layout-managers-mais-componentes-e-detalhes/>. Acesso em: 12 de maio de 2016.

Arquivo de códigos. Gerenciadores de layout: Apresentando o FlowLayout. Disponível em: <http://www.arquivodecodigos.net/arquivo/tutoriais/java/gerenciadores_layout_apresentando_flow_layout.php>. Acesso em: 13 de maio de 2016.

Arquivo de códigos. Gerenciadores de layout: Apresentando o BorderLayout. Disponível em: <http://www.arquivodecodigos.net/principal/diretorios/java/artigos_tutoriais/gerenciadores_layout_apresentando_border_layout.php>.Acesso em: 13 de maio de 2016.

Deitel. Java, Como Programar, 8ª Edição.