

[< VOLTAR](#)

Usando Gerenciadores de Layout

Construir interfaces gráficas manualmente em Java não é fácil, especialmente quando não se sabe qual a resolução do monitor de nosso cliente. Uma alternativa a isso é utilizar gerenciadores de layout, ou seja, funções prontas do Java que ajustam automaticamente os componentes dentro da janela, como o `FlowLayout`, `BorderLayout`, `GridLayout` e `GridBagLayout`. Veremos aqui a aplicação e exemplos de cada um destes gerenciadores de layout.

NESTE TÓPICO

- > Gerenciadores de layout em Java
- > `FlowLayout`
- > `BorderLayout`
- > `GridLayout`
- > `GridBagLayout`
- > Resumo
- > Referências



Gerenciadores de layout em Java



Utilizar os gerenciadores de layout pode ser uma boa ideia quando não se sabe a resolução da tela do(s) cliente(s) a qual sua aplicação será executada.

Até então utilizados o posicionamento e tamanho das telas de forma estática, ou seja, definindo nós mesmos os valores do tamanho e posição das telas. Contudo, segundo Teruel, essa forma de trabalho, muitas vezes, requer cálculos e demandam muito tempo de desenvolvimento.

Em Java temos mais de um tipo de gerenciador de layout, cada um com uma forma diferente de dividir a tela e administrar o espaço nela. Veremos, a seguir, exemplos de cada um deles e como cada um opera.

FlowLayout

Segundo Teruel, este é o gerenciador mais simples de layout, entre os outros. O *FlowLayout* faz o alinhamento padrão ao centro e acima, com deslocamento em linhas (*flow* = fluxo). Quando você adiciona o primeiro elemento, ele é posicionado ao centro e na parte de cima do contêiner. Quando você adiciona novos elementos, eles vão sendo posicionados em

linha, à direita do anterior. Se acabar a área disponível na linha, o elemento cai automaticamente para a próxima linha. Apesar dos elementos aparecerem por padrão ao centro, eles podem ser posicionados à esquerda ou à direita.

O *FlowLayout* também tem a capacidade de manter o tamanho original dos componentes durante o redimensionamento. Se mesmo se você redimensiona a janela, o tamanho dos componentes não é modificado, mantendo os valores originais.

Para exemplificar, vamos criar uma pequena tela utilizando o *FlowLayout*, com alguns componentes. Faremos uma tela com dois labels, dois campos de texto e dois botões. Usaremos duas classes para este novo projeto: A classe “Principal” (que, como sempre, contém o método main para ser executado pela primeira vez) e a classe “Tela1”, que usará o *FlowLayout*.

A classe “Tela1” será herdeira de JFrame. Sua codificação será a seguinte (veja atentamente cada uma das linhas da classe, pois algumas explicações estão nos comentários):

```

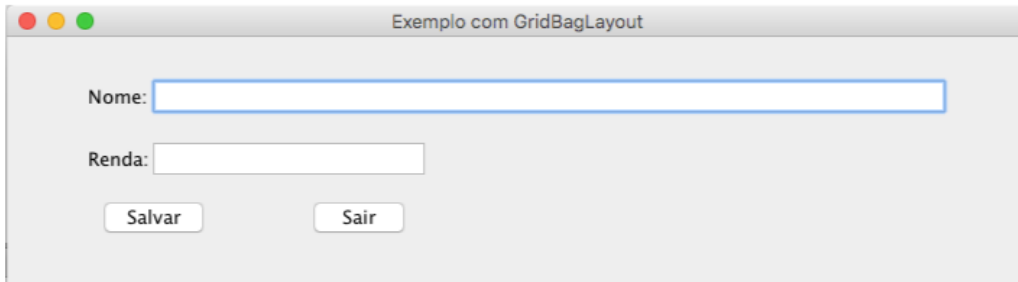
1. import java.awt.FlowLayout;
2. import javax.swing.JFrame;
3. import javax.swing.*;
4.
5. public class Tela1 extends JFrame {
6.
7.     //Propriedades da classe:
8.     private JLabel lblNome, lblRenda;
9.     private JTextField txtNome, txtRenda;
10.    private JButton btnSalvar, btnSair;
11.
12.    //Construtor da classe:
13.    public Tela1() {
14.        setTitle("Exemplo de Flow layout"); //Altera o título da janela
15.
16.        //Cria um objeto de FlowLayout:
17.        FlowLayout fl = new FlowLayout(); //Instancia
18.        fl.setAlignment(FlowLayout.RIGHT); //Seta o alinhamento
19.
20.        setLayout(fl); //Define que o gerenciador do layout será o Flowlayout criado
21.        setDefaultCloseOperation(EXIT_ON_CLOSE); //Define o que ocorre ao fechar a t
    ela
22.        setSize(650, 200); //Seta o tamanho da tela
23.        setLocation(100, 200); //Seta a posição da tela
24.
25.        //Cria os componentes e os adiciona
26.        lblNome = new JLabel("Nome:");
27.        add(lblNome);
28.        txtNome = new JTextField(45); //Para 45 caracteres
29.        add(txtNome);
30.        lblRenda = new JLabel("Renda:");
31.        add(lblRenda);
32.        txtRenda = new JTextField(15); //Para 15 caracteres
33.        add(txtRenda);
34.        btnSalvar = new JButton("Salvar");
35.        add(btnSalvar);
36.        btnSair = new JButton("Sair");
37.        add(btnSair);
38.    }
39. }
```



E nossa classe Principal será assim:

```
1. public class Principal {  
2.     public static void main(String args[]){  
3.         Tela1 t1 = new Tela1();  
4.         t1.setVisible(true);  
5.     }  
6. }
```

Veja o resultado da execução deste projeto:



Resultado da execução do projeto com FlowLayout

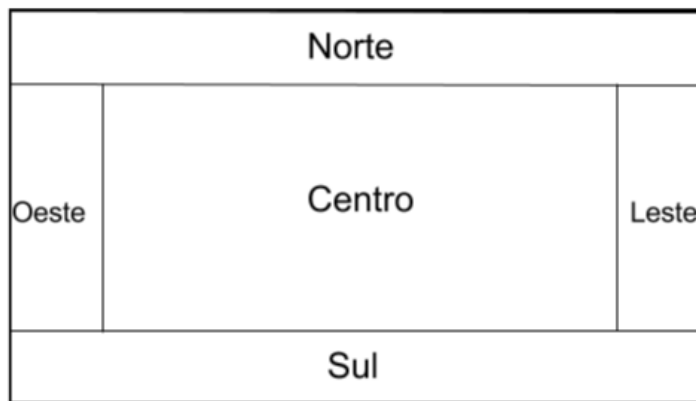
Note, na imagem acima, que os componentes estão alinhados à direita. Isso ocorre pois definimos o alinhamento no código da tela com a instrução "*setAlignment(FlowLayout.RIGHT)*". Você pode trocar o alinhamento para a esquerda (*FlowLayout.LEFT*) ou para o centro (*FlowLayout.CENTER*). Faça as alterações e veja a diferença.

Aumente manualmente a tela para a direita. Note que os componentes que estão na segunda linha sobem para a primeira, pois “sobre espaço”. Esse é o *FlowLayout* funcionando, ou seja, alinhando os componentes e os posicionando conforme há espaço na tela.



BorderLayout

Segundo Teruel, o gerenciador de layout *BorderLayout* divide automaticamente o contêiner em cinco grandes áreas: norte, sul, leste, oeste e centro. Além de não manter o tamanho original dos componentes durante o redimensionamento (já que eles são ampliados de acordo com a tela), o contêiner pode receber somente um componente por área. A Figura abaixo mostra como o gerenciador *BorderLayout* divide a janela em áreas:



Áreas do BorderLayout

Fonte: Adaptado de Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Lembre-se que o componente colocado em uma das áreas ocupará toda a área, automaticamente. Para ilustrar isso, vamos implementar uma tela (classe “Tela2”) com 5 botões, um para cada área. O código da “Tela2” pode ser visto abaixo:

```
1. import java.awt.BorderLayout;
2. import javax.swing.*;
3.
4. public class Tela2 extends JFrame {
5.
6.     //Atributos locais
7.     private JButton btnCentro, btnLeste, btnOeste, btnNorte, btnSul;
8.
9.     //Construtor:
10.    public Tela2() {
11.        setTitle("Exemplo com Border Layout");
12.        setLayout(new BorderLayout());
13.        setDefaultCloseOperation(EXIT_ON_CLOSE);
14.        setSize(700, 200);
15.        setLocation(300, 200);
16.        btnCentro = new JButton("Centro");
17.        add(btnCentro, BorderLayout.CENTER);
18.        btnLeste = new JButton("Leste");
19.        add(btnLeste, BorderLayout.EAST);
20.        btnSul = new JButton("Sul");
21.        add(btnSul, BorderLayout.SOUTH);
22.        btnNorte = new JButton("Norte");
23.        add(btnNorte, BorderLayout.NORTH);
24.        btnOeste = new JButton("Oeste");
25.        add(btnOeste, BorderLayout.WEST);
26.    }
27. }
```



Altere, agora, a classe principal para a seguinte codificação:

```
1. public class Principal {  
2.     public static void main(String args[]){  
3.         Tela2 t2 = new Tela2();  
4.         t2.setVisible(true);  
5.     }  
6. }
```

E o resultado da execução deste novo código será:



Execução do projeto com gerenciador de layout BorderLayout

Aumente e diminua a tela e veja que os botões mantêm a proporção da tela, ou seja, eles aumentam e diminuem junto com a tela. Isso é gerenciado pelo *BorderLayout*.



GridLayout

Segundo Teruel, o gerenciador de layout GridLayout divide o contêiner em linhas e colunas, similarmente a uma tabela com células (por isso o nome *grid*). Se você aumentar ou diminuir a janela, o tamanho dos componentes se manterão proporcionais às novas dimensões da janela. Em cada célula de nossa grid, ou seja, em cada posição do layout, você pode colocar apenas um elemento.

Para entendermos melhor, vamos programar uma pequena tela com os mesmos componentes: dois labels, dois campos de texto e dois botões, mas dessa vez com um componente em cada célula. Vamos implementar a classe ?Tela3?, agora, cujo código pode ser visto abaixo. Veja atentamente o código, pois as linhas estão comentadas com algumas explicações importantes:

```
1. import java.awt.GridLayout;
2. import javax.swing.*;
3.
4. public class Tela3 extends JFrame {
5.
6.     private JLabel lblNome, lblRenda;
7.     private JTextField txtNome, txtRenda;
8.     private JButton btnSalvar, btnSair;
9.
10.    //Construtor:
11.    public Tela3() {
12.
13.        setTitle("Exemplo com GridLayout"); //Titulo da tela
14.        setDefaultCloseOperation(EXIT_ON_CLOSE);
15.        setLayout(new GridLayout(3, 2)); //Numero de células (linhas X Colunas)
16.        setSize(600, 200); //tamanho da tela
17.        setLocation(300, 200); // POsição da tela
18.
19.        //Campos:
20.        lblNome = new JLabel("Nome:");
21.        add(lblNome);
22.        txtNome = new JTextField();
23.        add(txtNome);
24.        lblRenda = new JLabel("Renda:");
25.        add(lblRenda);
26.        txtRenda = new JTextField();
27.        add(txtRenda);
28.        btnSalvar = new JButton("Salvar");
29.        add(btnSalvar);
30.        btnSair = new JButton("Sair");
31.        add(btnSair);
32.    }
33. }
```

E podemos modificar nossa classe principal para:



```
1. public class Principal {
2.     public static void main(String args[]){
3.         Tela3 t3 = new Tela3();
4.         t3.setVisible(true);
5.     }
6. }
```

E o resultado dessa implementação será:



Resultado da execução do gerenciador GridLayout

É importante notar que no código definimos o tamanho da tabela. Neste caso 3x2 (linhas x colunas) e os componentes são inseridos em ordem nas células, ou seja, os componentes são inseridos da esquerda para a direita e

de cima para baixo, automaticamente pelo gerenciador.

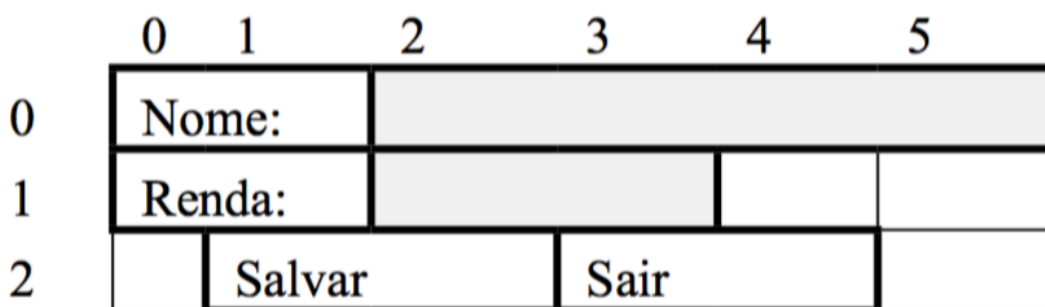
GridBagLayout

Segundo Teruel, o gerenciador de layout GridBagLayout é o mais flexível de todos, mas o mais complexo também, pois cada elemento deve conter uma posição inicial, uma posição final, tamanho, escala, alinhamento e um preenchimento.

Ele divide o contêiner em linhas e colunas similarmente a uma tabela com células (por isso o nome *grid*). Dependendo do alinhamento dentro de cada célula, pode-se ou não manter o tamanho original dos componentes lá inseridos. Em cada uma das células pode ser colocado apenas um único componente e cada célula pode ser expandida para ocupar o espaço de uma ou mais células vizinhas.

Quando utilizamos o GridBagLayout, um objeto da classe GridBagConstraints é, normalmente, utilizado para determinar o posicionamento dos elementos na *grid*.

Para entender melhor este conceito, vamos (re)desenvolver nosso primeiro exemplo, a tela que possui dois campos de textos (nome e renda) e dois botões, mas, vamos pensar nessa tela da seguinte maneira:



| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|--------|--------|------|---|---|---|
| 0 | Nome: | | | | | |
| 1 | Renda: | | | | | |
| 2 | | Salvar | Sair | | | |

Exemplo de layout GridBagLayout - Divisão em células

Fonte: Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Note que (Teruel):

- O rótulo (label) “Nome:” está na posição `gridx=0` e `gridy=0`, ocupando duas colunas (`gridwidth=2`).
- O campo para o preenchimento do nome está na posição `gridx=2` e `gridy=0`, ocupando quatro colunas (`gridwidth=4`).
- O rótulo (label) “Renda:” está na posição (`gridx=0` e `gridy=1`), ocupando duas colunas (`gridwidth=2`).
- O campo para o preenchimento da renda está na posição `gridx=2` e `gridy=1`, ocupando duas colunas (`gridwidth=2`).

- O botão “Salvar” está na posição (gridx=1 e gridy=2), ocupando duas colunas (gridwidth=2).
- O botão “Sair” está na posição (gridx=3 e gridy=2), ocupando duas colunas (gridwidth=2).

Sempre que for utilizar o gerenciador *GridBagLayout*, é muito importante planejar cuidadosamente o posicionamento dos componentes e quantos grids cada um irá ocupar, para que a distribuição seja uniforme e visualmente correta.

Para este exemplo, vamos usar uma nova classe, chamada “Tela4” (pode ser no mesmo projeto). O código dessa tela ficará assim (veja cuidadosamente o código, pois os comentários explicam o que ocorre em linhas importantes):




```
1. import java.awt.GridBagConstraints;
2. import java.awt.GridBagLayout;
3. import java.awt.Insets;
4. import javax.swing.JFrame;
5. import javax.swing.*;
6.
7. public class Tela4 extends JFrame {
8.
9.     //Propriedades locais:
10.    private JLabel lblNome, lblRenda;
11.    private JTextField txtNome, txtRenda;
12.    private JButton btnSalvar, btnSair;
13.
14.    public Tela4() {
15.        setTitle("Exemplo com GridBagLayout"); //Titulo da janela
16.        setDefaultCloseOperation(EXIT_ON_CLOSE); //O que acontece quando fechar a ja
ne;a
17.        setSize(700, 200); //Tamanho da janela
18.        setLayout(new GridBagLayout()); //Seta o gerenciador que será usado
19.        GridBagConstraints gridCons = new GridBagConstraints();
20.
21.        //Seta as propriedades dos parâmetros locais:
22.        lblNome = new JLabel("Nome:");
23.        txtNome = new JTextField(45);
24.        lblRenda = new JLabel("Renda:");
25.        txtRenda = new JTextField(15);
26.        btnSalvar = new JButton("Salvar");
27.        btnSair = new JButton("Sair");
28.
29.        //Seta a grid e posiciona os elementos:
30.        gridCons.gridx = 0;
31.        gridCons.gridy = 0;
32.        gridCons.gridwidth = 2;
33.        add(lblNome, gridCons);
34.
35.        gridCons.gridx = 2;
36.        gridCons.gridy = 0;
37.        gridCons.gridwidth = 4;
38.        add(txtNome, gridCons);
39.
40.        gridCons.gridx = 0;
41.        gridCons.gridy = 1;
42.        gridCons.gridwidth = 2;
43.        gridCons.insets = new Insets(15, 0, 8, 0);
44.        add(lblRenda, gridCons);
45.
46.        gridCons.gridx = 2;
47.        gridCons.gridy = 1;
48.        gridCons.gridwidth = 2;
49.        gridCons.anchor = GridBagConstraints.WEST;
50.        add(txtRenda, gridCons);
51.
52.        gridCons.gridx = 1;
53.        gridCons.gridy = 2;
54.        gridCons.gridwidth = 2;
55.        gridCons.insets = new Insets(5, 5, 5, 5);
56.        gridCons.anchor = GridBagConstraints.CENTER;
57.        add(btnSalvar, gridCons);
58.
59.        gridCons.gridx = 3;
60.        gridCons.gridy = 2;
61.        gridCons.gridwidth = 2;
62.        gridCons.anchor = GridBagConstraints.CENTER;
63.        add(btnSair, gridCons);
64.    }
65. }
```



E modificamos a nossa classe principal para:

```
1. public class Principal {  
2.     public static void main(String args[]){  
3.         Tela4 t4 = new Tela4();  
4.         t4.setVisible(true);  
5.     }  
6. }
```

E, finalmente, o resultado da nova execução do projeto será:



Execução do projeto considerando o gerenciador GridBagLayout

Resumo

Chegamos ao final de mais um importante conceito envolvendo interfaces gráficas, utilizando gerenciadores de layout que cuidam do posicionamento e administração visual dos componentes em nossas telas. Vimos, aqui, os principais gerenciadores, que são:

- FlowLayout
- BorderLayout
- GridLayout
- GridBagLayout

De qualquer forma, mesmo utilizando os gerenciadores de layout, é preciso planejar cuidadosamente suas interfaces, para que fiquem agradáveis ao serem utilizadas e proporcionem uma boa experiência ao usuário.

Pratique bastante os códigos acima e faça modificações livremente para ver o comportamento de cada um dos gerenciadores. Boa programação e bom estudo!

Quiz

Exercício Final

Usando Gerenciadores de Layout

INICIAR ➤

Referências

Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Deitei P. e Deitel H., 2010, Java : Como programar, 8ª Edição, Pearson Pretice Hall

Schildt, H., 2015, Schildt, Java para iniciantes : crie, compile e execute programas Java rapidamente, Bookman



Avalie este tópico



ANTERIOR

Lidando com Eventos de GUI em Java

Biblioteca

(https://www.uninove.br/conheca-

a-

uninove/biblioteca/sobre-

a-

biblioteca/apresentacao/)

Portal Uninove

(http://www.uninove.br)

Mapa do Site



Índice

Ajuda?

(https://ava.un

idCurso=)

Construção de GUI em Java

® Todos os direitos reservados

