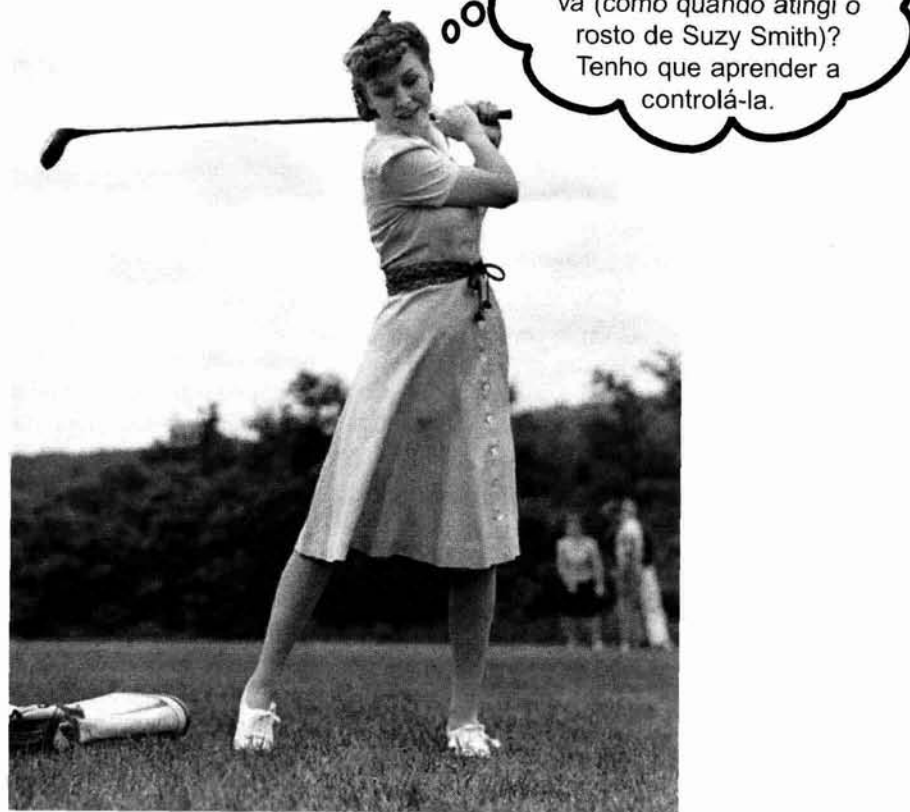


Trabalhe em Seu Swing



Por que a bola não vai para onde quero que ela vá (como quando atingi o rosto de Suzy Smith)? Tenho que aprender a controlá-la.

O Swing é fácil. A menos que você se *importe* realmente com o local onde as coisas acabarão ficando na tela. O código Swing *parece* fácil, mas, depois de compilar, executar e examiná-lo nos damos conta “ei, *isso* não deveria estar *ai*”. O que torna *fácil* a *codificação* é o que torna *difícil* o *controle* - o **Gerenciador de Layout**. Os objetos do Gerenciador de Layout controlam o tamanho e o local dos elementos gráficos em uma GUI Java. Eles executarão várias tarefas por você, que nem sempre gostará dos resultados. Você pode querer dois botões do mesmo tamanho, o que eles não terão. Pode querer que o campo de texto tenha três polegadas, mas ele terá nove. Ou uma. E *abaixo* do rótulo em vez de *ao lado* dele. Mas, com um pouco de esforço, você pode fazer os gerenciadores de layout se curvarem à sua vontade. Neste capítulo, trabalharemos em nosso Swing e, além dos gerenciadores de layout, aprenderemos mais sobre os elementos gráficos. Criaremos, exibiremos (onde quisermos) e os usaremos em um programa. Não está parecendo muito bom para Suzy.

Componentes do Swing

Componente é o termo mais correto para o que temos chamado de *elemento gráfico*. As coisas que você vai inserir em uma GUI. As coisas que um usuário verá e com as quais interagirá. Campos de texto, botões, listas roláveis, botões de rádio, etc. são todos componentes. Na verdade, todos estendem `javax.swing.JComponent`.

Os componentes podem ser aninhados


No Swing, praticamente *todos* os componentes podem conter outros componentes. Em outras palavras, *você pode inserir quase tudo em qualquer outra coisa*. Mas, na maioria das situações, você adicionará componentes de interação com o usuário como botões e listas em componentes de plano de fundo como molduras e painéis. Embora seja possível inserir, digamos, um painel dentro de um botão, isso seria muito estranho, e não lhe renderá nenhum prêmio de aproveitamento.

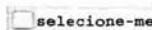
Com exceção de `JFrame`, no entanto, a diferença entre componentes interativos e componentes de plano de fundo é artificial. Um `JPanel`, por exemplo, geralmente é usado como o plano de fundo para o agrupamento de outros componentes, mas até esse componente pode ser interativo. Exatamente como ocorre com os outros componentes, você pode se registrar para ouvir eventos de `JPanel`, inclusive cliques no mouse e pressionamento de teclas.

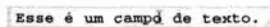
Quatro etapas para a criação de uma GUI (revisão)

- ① **Crie uma janela (um `JFrame`).**
`JFrame frame = new JFrame();`
- ② **Crie um componente (botão, campo de texto, etc.).**
 `JButton button = new JButton("click me");`
- ③ **Adicione o componente à moldura.**
`frame.getContentPane().add(BorderLayout.EAST, button);`
- ④ **Exiba-o (forneça um tamanho e torne-o visível).**
`frame.setSize(300,300);`
`frame.setVisible(true);`

Insira componentes interativos:

 `JButton`

 `JCheckBox`

 `JTextField`

Em componentes de plano de fundo:



`JFrame`



`JPanel`

Um elemento gráfico é tecnicamente um Componente do Swing.

Quase tudo que você inserir em uma GUI estenderá `javax.swing.JComponent`.

Gerenciadores de layout

Um gerenciador de layout é um objeto Java associado a um componente específico, quase sempre um componente de plano de fundo. O gerenciador de layout controla os componentes que se encontram dentro do componente ao qual ele está associado. Em outras palavras, se uma moldura tiver um painel, e o painel tiver um botão, o gerenciador de layout do painel controlará o tamanho e a inserção do botão, enquanto o gerenciador de layout da moldura controlará o tamanho e a inserção do painel. O botão, por outro lado, não precisa de um gerenciador de layout, porque não contém outros componentes.

Se um painel tiver cinco elementos, mesmo se cada um desses cinco elementos tiver seus próprios gerenciadores de layout, seu tamanho e local no painel serão controlados pelo gerenciador de layout do painel. Se, por sua vez, esses cinco elementos tiverem outros elementos, então, esses outros elementos serão inseridos de acordo com o gerenciador de layout do elemento que os contém.

Quando dizemos *conter*, queremos na verdade dizer *adicionar* como em, um painel *contém* um botão porque o botão foi *adicionado* a ele através de algo como:

```
myPanel.add(button);
```

Os gerenciadores de layout vêm em várias versões, e cada componente de plano de fundo pode ter seu próprio gerenciador de layout. Os gerenciadores de layout têm suas próprias políticas a seguir quando constroem um layout. Por exemplo, um gerenciador de layout pode insistir que todos os componentes de um painel tenham o mesmo tamanho, organizados em uma grade, enquanto outro gerenciador pode permitir que cada componente tenha seu próprio tamanho, contanto que fiquem empilhados verticalmente. Aqui está um exemplo de layouts aninhados:

```
JPanel panelA = new JPanel();

JPanel panelB = new JPanel();

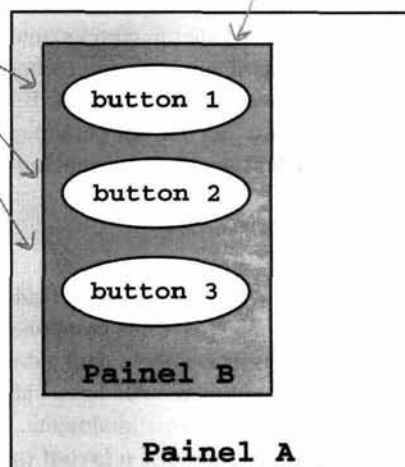
panelB.add(new JButton("button 1"));
panelB.add(new JButton("button 2"));
panelB.add(new JButton("button 3"));

panelA.add(panelB);
```



O gerenciador de layout do painel B controla o tamanho e a inserção dos três botões.

O gerenciador de layout do painel A controla o tamanho e a inserção do Painel B.



O gerenciador de layout do painel A não tem NADA a dizer quanto aos três botões. A hierarquia de controle só tem um nível – o gerenciador de layout do painel A controlará apenas os elementos adicionados diretamente a esse painel e não controlará nada que esteja aninhado dentro dos componentes que foram adicionados.

Como o gerenciador de layout decide?

Diferentes gerenciadores de layout têm políticas distintas para a organização de componentes (como organizar em uma grade, fazer com que todos tenham o mesmo tamanho, empilhá-los verticalmente, etc.), mas os componentes que estiverem sendo dispostos terão pelo menos *alguma* pequena influência na questão. Geralmente, o processo de dispor um componente de plano de fundo é semelhante ao descrito a seguir:

Um cenário de layout:

- ① Crie um painel e adicione três botões a ele.
- ② O gerenciador de layout do painel perguntará a cada botão que tamanho ele prefere ter.
- ③ O gerenciador de layout do painel usará suas políticas de layout para decidir se deve respeitar todas, parte ou nenhuma das preferências dos botões.
- ④ Adicione o painel a uma moldura.
- ⑤ O gerenciador de layout da moldura perguntará ao painel o tamanho que ele prefere ter.
- ⑥ O gerenciador de layout da moldura usará suas políticas de layout para decidir se deve respeitar todas, parte ou nenhuma das preferências do painel.

Diferentes gerenciadores de layout têm características distintas

Alguns gerenciadores de layout respeitam o tamanho que o componente quer ter. Se o botão quiser ter 30 por 50 pixels, será isso que o gerenciador de layout alocará para ele. Outros gerenciadores de layout respeitam somente parte do tamanho preferido pelo componente. Se o botão quiser ter 30 por 50 pixels, ele terá 50 pixels e a largura que seu *painel* de plano de fundo tiver. Outros respeitam somente a preferência do *maior* entre os componentes que estão sendo dispostos, e os demais componentes desse painel serão todos criados com esse mesmo tamanho. Em alguns casos, a tarefa do gerenciador de layout pode se tornar muito complexa, mas quase sempre você conseguirá descobrir o que provavelmente ele fará, quando conhecer as características desse gerenciador de layout.

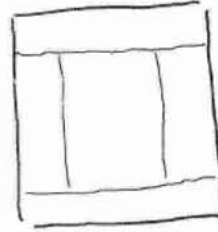


gerenciador de layout

Os três grandes gerenciadores de layout: limite, fluxo e caixa

BorderLayout

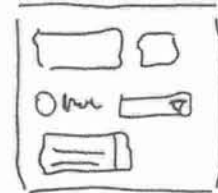
Um gerenciador BorderLayout divide um componente de plano de fundo em cinco regiões. Você só poderá adicionar um componente por região a um plano de fundo controlado por um gerenciador BorderLayout. Os componentes dispostos por esse gerenciador geralmente não conseguem ter seu tamanho preferido. **BorderLayout é o gerenciador de layout padrão para uma moldura!**



um componente por região

FlowLayout

Um gerenciador FlowLayout age como um processador de palavras, porém com componentes em vez de palavras. Cada componente recebe o tamanho que deseja, e eles são dispostos da esquerda para a direita na ordem que são adicionados, com a “mudança automática de linha” ativada. Portanto, quando um componente não couber horizontalmente, ele passará para a “linha” seguinte do layout. **FlowLayout é o layout padrão para um painel!**



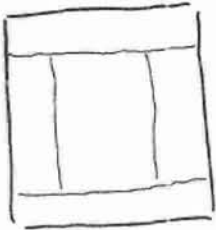
Componentes são adicionados da esquerda para a direita, passando para uma nova linha quando necessário

BoxLayout

Um gerenciador BoxLayout é como FlowLayout pelo fato de cada componente poder ter seu próprio tamanho e pelos componentes serem inseridos na ordem em que são adicionados. Mas, diferente de FlowLayout, um gerenciador BoxLayout pode empilhar os componentes verticalmente (ou horizontalmente, mas em geral só nos preocupamos com a disposição vertical). É como FlowLayout, mas em vez de ter a ‘mudança do componente para outra linha’ automaticamente, você poderá inserir um tipo de ‘tecla return de componentes’ e forçá-los a começar em uma nova linha.



Componentes adicionados de cima para baixo, um por ‘linha’.



**BorderLayout leva em consideração cinco regiões:
leste, oeste, norte, sul e centro**

Adicionaremos um botão à região leste:

```
import javax.swing.*;
import java.awt.*;

public class Button1 {

    public static void main (String[] args) {
        Button1 gui = new Button1();
        gui.go();
    }

    public void go() {
        JFrame frame = new JFrame();
        JButton button = new JButton("click me");
        frame.getContentPane().add(BorderLayout.EAST, button);
        frame.setSize(200,200);
        frame.setVisible(true);
    }
}
```

BorderLayout
está no pacote
java.awt

especifica a
região

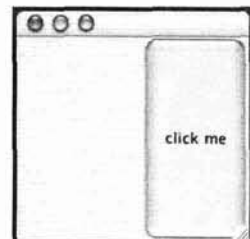


Halterofilismo cerebral

Como o gerenciador BorderLayout definiu esse tamanho para o botão?

Quais são os fatores que o gerenciador de layout tem que considerar?

Por que o botão não ficou mais largo ou mais alto?



Veja o que acontece ao fornecermos mais caracteres para o botão...

```

public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton("click like you mean it");
    frame.getContentPane().add(BorderLayout.EAST, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}

```

Alteramos somente
o texto do botão.

Primeiro,
perguntarei ao
botão seu
tamanho
preferido.

Agora tenho
várias palavras,
portanto, gostaria
de ter 60 pixels
de largura e 25
de altura.



Já que ele está na região leste de um
gerenciador de limites, respeitarei sua
largura preferida. Mas não me importa
a altura que ele deseja ter; terá a
mesma altura da moldura, porque essa
é minha política.

Da próxima vez
vou usar o layout
de fluxo. Assim
poderei fazer o
que quiser.



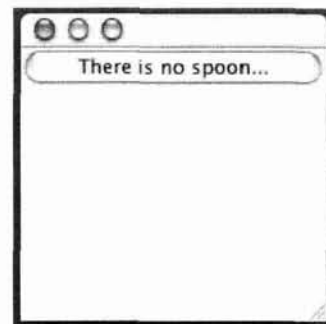
O botão ficou com a largura
solicitada, mas não a altura.

Tentaremos inserir um botão na região norte

```

public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton(
        "There is no spoon...");
    frame.getContentPane().add(
        BorderLayout.NORTH, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}

```



O botão ficou com sua altura
preferida, mas com a largura da
moldura.

Agora façamos o botão pedir para ser mais alto

Como fazer isso? O botão já está com o máximo da largura que pode ter — a largura da moldura. Mas podemos tentar torná-lo mais alto fornecendo uma fonte maior.

```

public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton(
        "Click This!");
    Font bigFont = new Font("serif",
        Font.BOLD, 28);
    button.setFont(bigFont);
    frame.getContentPane().add(
        BorderLayout.NORTH, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}

```

Uma fonte maior forçará a
moldura a alocar mais espaço
para a altura do botão.



A largura continuou a mesma, mas agora o botão
está mais alto. A região norte foi esticada para
acomodar a nova altura preferida do botão.

Acho que estou entendendo... Se eu estiver na região **leste** ou **oeste**, ficarei com minha largura preferida, mas a altura será definida pelo gerenciador de layout. E se eu estiver na região **norte** ou **sul**, ocorrerá o oposto — ficarei com minha altura preferida, mas não a largura.



Mas o que acontece na região central?

A região central fica com o que sobrar!

(Exceto em um caso especial que examinaremos posteriormente.)

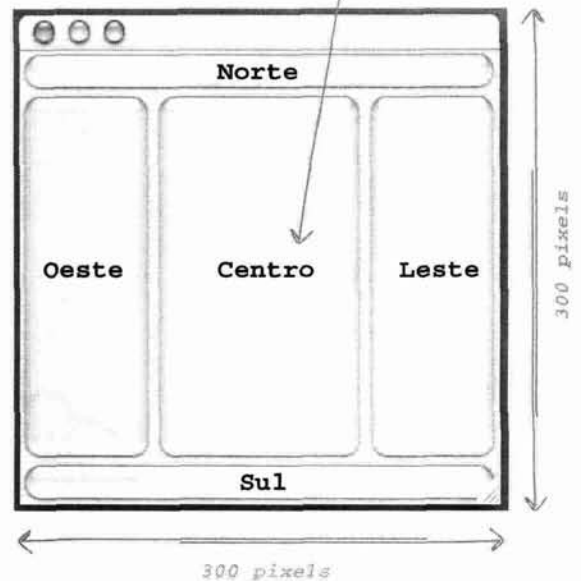
```
public void go() {
    JFrame frame = new JFrame();

    JButton east = new JButton("East");
    JButton west = new JButton("West");
    JButton north = new JButton("North");
    JButton south = new JButton("South");
    JButton center = new JButton("Center");

    frame.getContentPane().add(BorderLayout.EAST, east);
    frame.getContentPane().add(BorderLayout.WEST, west);
    frame.getContentPane().add(BorderLayout.NORTH, north);
    frame.getContentPane().add(BorderLayout.SOUTH, south);
    frame.getContentPane().add(BorderLayout.CENTER, center);

    frame.setSize(300,300);
    frame.setVisible(true);
}
```

Os componentes do centro ficarão com o espaço que sobrar, de acordo com as dimensões da moldura (300 x 300 nesse código).



Os componentes da região leste e oeste ficarão com sua largura preferida.

Os componentes da região norte e sul ficarão com sua altura preferida.

Quando você inserir alguma coisa na região norte ou sul, ela se estenderá de um lado a outro da moldura, portanto os elementos das regiões leste e oeste não ficarão tão altos quanto seriam se as regiões norte e sul estivessem vazias.



FlowLayout leva em consideração o fluxo dos componentes: da esquerda para a direita, na ordem em que foram adicionados.

Adicionemos um painel à região leste:

O gerenciador de layout de um objeto JPanel é FlowLayout, por padrão. Quando adicionarmos um painel a uma moldura, seu tamanho e inserção ainda estarão sob o controle do gerenciador BorderLayout. Mas qualquer coisa que estiver *dentro* do painel [em outras palavras, os componentes adicionados ao painel pela chamada a **panel.add(aComponent)**] estarão sob o controle de seu gerenciador FlowLayout. Começaremos inserindo um painel vazio na região leste da moldura e na próxima página adicionaremos elementos ao painel.

```
import javax.swing.*;
import java.awt.*;

public class Panell {

    public static void main (String[] args) {
        Panell gui = new Panell();
        gui.go();
    }
}
```



O painel não tem nada nele, portanto, não precisa ser muito largo na região leste.

```

public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);
    frame.getContentPane().add(BorderLayout.EAST, panel);
    frame.setSize(200,200);
    frame.setVisible(true);
}

```

Torna o painel cinza para podermos ver onde ele está na moldura.

Adicionemos um botão ao painel

```

public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);

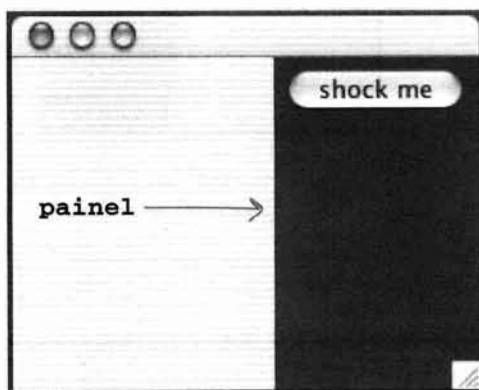
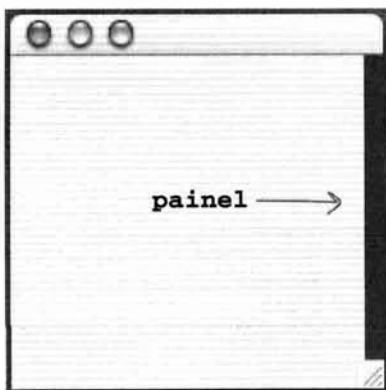
    JButton button = new JButton("shock me");

    panel.add(button);
    frame.getContentPane().add(BorderLayout.EAST, panel);

    frame.setSize(250,200);
    frame.setVisible(true);
}

```

Adiciona o botão ao painel e o painel à moldura. O gerenciador de layout do painel (de fluxo) controla o botão e o gerenciador de layout da moldura (de limite) controla o painel.



O painel se expandiu! E o botão ficou com seu tamanho preferido nas duas dimensões, porque o painel usa o layout de fluxo e o botão faz parte do painel (e não da moldura).

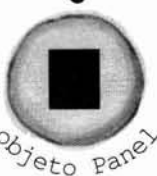
Certo... Preciso saber que tamanho o **painel** quer ter...



O gerenciador
BorderLayout da moldura

Tenho um botão agora, portanto, meu gerenciador de layout terá que descobrir o tamanho que preciso ter...

controla



objeto Panel

Preciso saber que tamanho o **botão** quer



O gerenciador
FlowLayout do painel

Com base no tamanho de minha fonte e na quantidade de caracteres, quero ter 70 pixels de largura e 20 de altura.

controla



objeto Button

O que acontecerá se adicionarmos DOIS botões ao painel?

```
public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);

    JButton button = new JButton("shock me");
    JButton buttonTwo = new JButton("bliss");

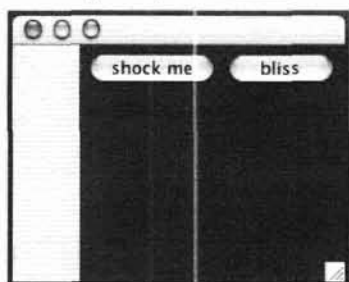
    panel.add(button);
    panel.add(buttonTwo);

    frame.getContentPane().add(BorderLayout.EAST, panel);
    frame.setSize(250,200);
    frame.setVisible(true);
}
```

cria DOIS botões

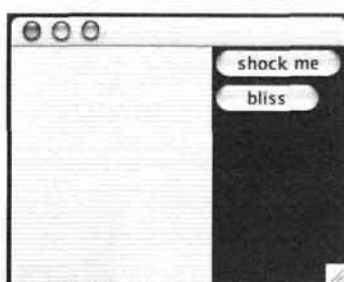
adiciona os DOIS ao painel

o que queríamos



Queremos os botões empilhados um acima do outro

o que obtivemos



O painel se expandiu para inserir os dois botões lado a lado.

Observe que o botão 'bliss' é menor do que o botão 'shock me'... É assim que o layout de fluxo funciona. O botão obtém exatamente o que precisa (e nada mais).

Aponte seu lápis

Se o código anterior fosse alterado para o código a seguir, qual seria a aparência da GUI?

```
JButton button = new JButton("shock me");
JButton buttonTwo = new JButton("bliss");
JButton buttonThree = new JButton("huh?");
panel.add(button);
panel.add(buttonTwo);
panel.add(buttonThree);
```



Desenhe qual acha que seria a aparência da GUI se você executasse o código à esquerda. (Em seguida, teste-o!)



BoxLayout vem nos salvar!

Ele manterá os componentes empilhados, mesmo se houver espaço para inseri-los lado a lado.

Diferente de FlowLayout, BoxLayout pode forçar a criação de uma 'nova linha' para fazer os componentes passarem para a linha seguinte, mesmo se houver espaço para eles se ajustarem horizontalmente.

Mas agora você terá que alterar o gerenciador de layout do painel do FlowLayout padrão para BoxLayout.

```
public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);

    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

    JButton button = new JButton("shock me");
    JButton buttonTwo = new JButton("bliss");
    panel.add(button);
    panel.add(buttonTwo);
    frame.getContentPane().add(BorderLayout.EAST, panel);
    frame.setSize(250,200);
    frame.setVisible(true);
}
```

Altera o gerenciador de layout para que seja uma nova instância de BoxLayout.

O construtor de BoxLayout precisa conhecer o componente que está dispondo (isto é, o painel) e que eixo usar (usaremos Y_AXIS para um empilhamento vertical).

Observe como o painel está mais estreito novamente, porque não precisa ajustar os dois botões horizontalmente. Portanto, ele informou à moldura que precisava de espaço suficiente somente para o botão principal, 'shock me'.



Não existem Perguntas Idiotas

P: Por que você não pode adicionar algo diretamente a uma moldura como podemos fazer em um painel?

R: Um JFrame é especial porque é onde a ação realmente ocorre quando queremos fazer com que algo apareça na tela. Embora todos os componentes do Swing sejam Java puro, um JFrame tem que se conectar ao sistema operacional subjacente para acessar a exibição. Considere o painel de conteúdo como uma camada com 100% de Java puro que fica *acima* do JFrame. Ou considere como se o JFrame fosse a moldura da janela e o painel de conteúdo fosse a... Vidraça. Você sabe, a *vidraça* da janela. E você pode até *trocar* o painel de conteúdo pelo seu próprio JPanel, para tornar seu JPanel o painel de conteúdo da moldura, usando:

```
myFrame.setContentPane(myPanel);
```

P: Posso alterar o gerenciador de layout da moldura? E se eu quiser que a moldura use o gerenciador de fluxo em vez do gerenciador de limite?

R: A maneira mais fácil de fazer isso é criar um painel, construir a GUI como você quiser no painel e, em seguida, tornar esse painel o painel de conteúdo da moldura usando o código da resposta anterior (em vez de usar o painel de conteúdo padrão).

P: E se eu quiser um tamanho preferido diferente? Há um método `setSize()` para componentes?

R: Sim, há um `setSize()`, mas os gerenciadores de layout o ignorarão. Há uma diferença entre o *tamanho preferido* do componente e o tamanho que *você* quer que ele tenha. O tamanho preferido é baseado no tamanho que o componente realmente *precisa* ter (o componente toma essa decisão sozinho). O gerenciador de layout chamará o método `getPreferredSize()` do componente e esse método não se *importará* se antes você chamou `setSize()` no componente.

P: Não posso simplesmente inserir os elementos onde quiser? Posso desativar os gerenciadores de layout?

R: Sim. Em cada componente isoladamente, você pode chamar `setLayout(null)`, e assim ficará sob sua responsabilidade embutir em código os locais e dimensões exatos da tela. No final das contas, no entanto, quase sempre é mais fácil usar os gerenciadores de layout.

DISCRIMINAÇÃO DOS PONTOS

- Os gerenciadores de layout controlam o tamanho e o local de componentes aninhados dentro de outros componentes.

- Quando você adicionar um componente a outro componente (às vezes chamado de componente *de plano de fundo*, mas essa não é uma classificação técnica), o componente adicionado será controlado pelo gerenciador de layout do componente *de plano de fundo*.

- Um gerenciador de layout pergunta aos componentes seu tamanho preferido, antes de tomar uma decisão sobre o layout. Dependendo das políticas do gerenciador de layout, ele pode respeitar todas, algumas ou nenhuma das preferências do componente.

- O gerenciador `BorderLayout` permitirá que você adicione um componente a uma das cinco regiões. Você deve especificar a região quando adicionar o componente, usando a sintaxe a seguir:

```
add(BorderLayout.EAST, panel);
```

- Com `BorderLayout`, os componentes das regiões norte e sul ficam com sua altura preferida, mas não com a largura. Os componentes das regiões leste e oeste ficam com sua largura preferida, mas não a altura. O componente do centro ficará com o que sobrar [a menos que você use `pack()`].

- O método `pack()` é como se disséssemos 'embalado e lacrado' com relação aos componentes; ele usa o tamanho preferido total do componente central e, em seguida, determina o tamanho da moldura, usando o centro como ponto inicial, construindo o resto com base no que houver nas outras regiões.

- `FlowLayout` insere os componentes da esquerda para a direita, de cima para baixo, na ordem que foram adicionados, passando para um nova linha de componentes somente quando eles não cabem horizontalmente.

- `FlowLayout` dá aos componentes seu tamanho preferido nas duas dimensões.

- `BoxLayout` permitirá que você alinhe os componentes empilhados verticalmente, mesmo se eles couberem lado a lado. Como `FlowLayout`, `BoxLayout` usa o tamanho preferido do componente nas duas dimensões.

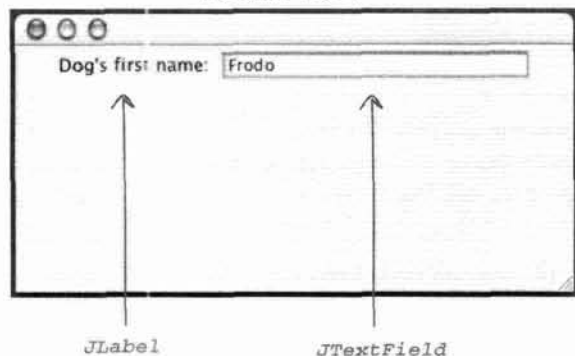
- `BorderLayout` é o gerenciador de layout padrão para uma moldura; `FlowLayout` é o padrão para um painel.

- Se você quiser que um painel use algo diferente do fluxo, terá que chamar `setLayout()` no painel.

Testando os componentes do Swing

Você aprendeu os aspectos básicos dos gerenciadores de layout, portanto agora testaremos alguns dos componentes mais comuns: um campo de texto, a área de texto de rolagem, a caixa de seleção e a lista. Não mostraremos o API inteiro de cada um, apenas alguns destaques como introdução.

JTextField



Construtores:

```
JTextField field = new JTextField(20);
```

20 significa 20 colunas e não 20 pixels.
Isso define a largura preferida do campo de texto.

```
JTextField field = new JTextField("Your name");
```

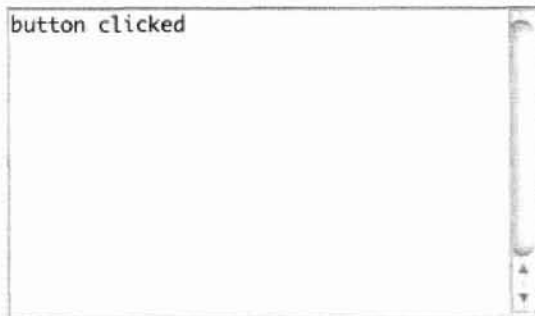
Como usá-lo

- ① Capture o texto que se encontra nele.
`System.out.println(field.getText());`
- ② Insira texto nele.
`field.setText("whatever");`
`field.setText("");`
Isso limpa o campo.
- ③ Capture um `ActionEvent` quando o usuário pressionar `return` ou `enter`.
`field.addActionListener(myActionListener);`

Você também pode se registrar para ouvir eventos-chave, se quiser realmente ser informado sempre que o usuário pressionar uma tecla.

- ④ Selecione/realce o texto do campo.
`field.selectAll();`
- ⑤ Posicione o cursor no campo (para que o usuário possa começar a digitar).
`field.requestFocus();`

JTextArea



Diferente de `JTextField`, `JTextArea` pode ter mais de uma linha de texto. É preciso algum esforço de configuração em sua criação, porque ele não vem com barras de rolagem ou quebra de linha. Para fazer um `JTextArea` rolar, você terá que inseri-lo em um `ScrollPane`. Um `ScrollPane` é um objeto que aprecia muito rolar e se encarregará das necessidades de rolagem da área de texto.

Construtores:

10 significa 10 linhas (configura a altura preferida).

```
JTextArea text = new JTextArea(10,20);
```

20 significa 20 colunas (configura a largura preferida).

Como usá-lo

Faça com que ele tenha somente uma barra de rolagem vertical.

```
JScrollPane scroller = new JScrollPane(text);
```

Cria um `JScrollPane` e lhe fornece a área de texto para a qual ele rolará.

```
text.setLineWrap(true);
```

```
scroller.setVerticalScrollBarPolicy  
    (ScrollPaneConstants.  
        VERTICAL_SCROLLBAR_ALWAYS);  
scroller.setHorizontalScrollBarPolicy  
    (ScrollPaneConstants.  
        HORIZONTAL_SCROLLBAR_NEVER);
```

Informa ao painel de rolagem para usar somente uma barra de rolagem vertical.

```
panel.add(scroller);
```

Importante! Você fornecerá a área de texto ao painel de rolagem (através do construtor do painel de rolagem) e, em seguida, adicionará o painel de rolagem ao painel geral. Você não adicionará a área de texto diretamente ao painel geral!

Substitua o texto existente.

```
text.setText("Not all who are lost are  
wandering");
```

Acrescente algo ao texto existente.
`text.append("button clicked");`

- ④ Selecione/realce o texto do campo.
`text.selectAll();`
- ⑤ Posicione o cursor no campo (para que o usuário possa começar a digitar).
`text.requestFocus();`

Exemplo de JTextArea

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TextAreal implements ActionListener {

    JTextArea text;

    public static void main (String[] args) {
        TextAreal gui = new TextAreal();
        gui.go();
    }

    public void go() {
        JFrame frame = new JFrame();
        JPanel panel = new JPanel();
        JButton button = new JButton("Just Click It");
        button.addActionListener(this);
        text = new JTextArea(10,20);
        text.setLineWrap(true);

        JScrollPane scroller = new JScrollPane(text);
        scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

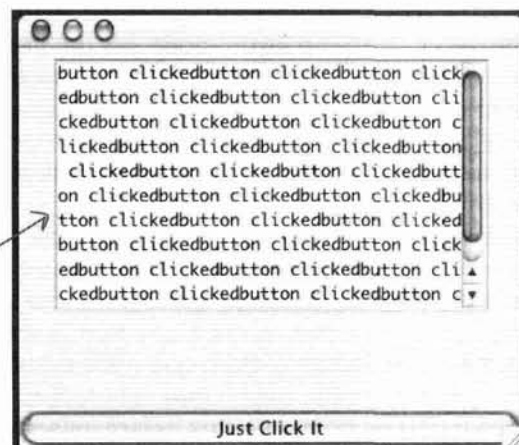
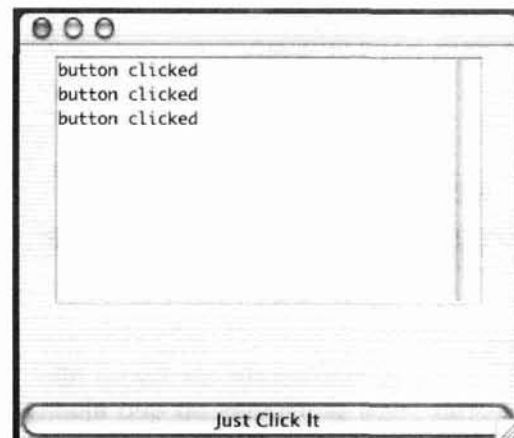
        panel.add(scroller);

        frame.getContentPane().add(BorderLayout.CENTER, panel);
        frame.getContentPane().add(BorderLayout.SOUTH, button);

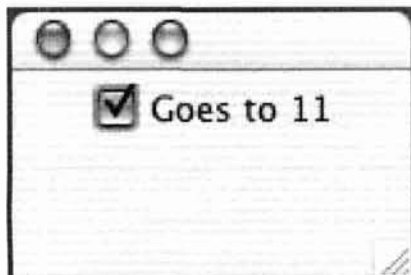
        frame.setSize(350,300);
        frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent ev) {
        text.append("button clicked \n ");
    }
}
```

Insere uma nova linha para que as palavras sejam inseridas em uma linha separada sempre que o botão for clicado. Caso contrário, ficará tudo junto.



JCheckBox



Construtores:

```
JCheckBox check = new JCheckBox("Goes to 11");
```

Como usá-lo

- ① Escute o evento de um item (quando ele for selecionado ou desmarcado).

```
check.addItemListener(this);
```
- ② Manipule o evento (e descubra se ele foi ou não selecionado).

```
public void itemStateChanged(ItemEvent ev) {
    String onOrOff = "off";
    if (check.isSelected()) onOrOff = "on";
    System.out.println("Check box is " + onOrOff);
}
```
- ③ Selecione ou desmarque-o em código.

```
check.setSelected(true);
check.setSelected(false);
```

Não existem Perguntas Idiotas

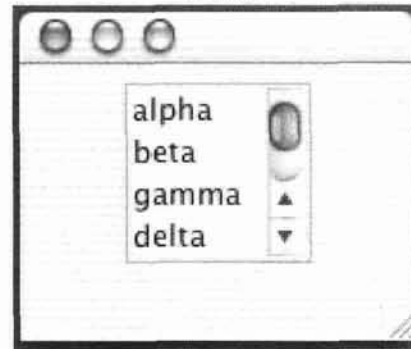
P: Os gerenciadores de layout não causam mais problemas do que ajudam? Se eu tiver que passar por toda essa confusão, prefiro embutir em código o tamanho e as coordenadas de onde os elementos devem ser inseridos.

R: Obter o layout exato que você deseja de um gerenciador de layout pode ser um desafio. Mas pense no que o gerenciador de layout está realmente fazendo para você. Mesmo a tarefa aparentemente simples de saber onde os elementos devem ser inseridos na tela pode ser complexa. Por exemplo, o gerenciador de layout se encarregará de evitar que seus componentes se sobreponham. Em outras palavras, ele sabe como gerenciar o espaçamento entre os componentes (e entre a borda da moldura). É claro que você pode fazer isso sozinho, mas o que acontecerá se quiser que os componentes sejam inseridos bem próximos. Você pode conseguir inseri-los da maneira correta, manualmente, mas isso só será bom para sua JVM!

Por quê? Porque os componentes podem ser um pouco diferentes de uma plataforma para outra, principalmente se usarem a 'aparência' nativa da plataforma subjacente. Coisas sutis como o contorno dos botões podem ser tão diferentes que os componentes que ficam alinhados corretamente em uma plataforma repentinamente se amontoam em outra.

E ainda não chegamos ao que os gerenciadores fazem de realmente importante. Pense no que acontecerá quando o usuário redimensionar a janela! Ou se sua GUI for dinâmica, onde componentes surgem e desaparecem. Se você tivesse que controlar a reorganização de todos os componentes sempre que houvesse uma alteração no tamanho ou no conteúdo de um componente de plano de fundo... É bom nem pensar!

JList



Construtores:

```
String [] listEntries = {"alpha", "beta", "gamma",
                        "delta", "epsilon", "zeta", "eta", "theta"}

list = new JList(listEntries);
```

O construtor de JList usa uma matriz de qualquer tipo de objeto. Eles não têm que ser Strings, mas a representação de uma String aparecerá na lista.

Como usá-lo

- ① **Faça com que ele tenha uma barra de rolagem vertical.**

```
JScrollPane scroller = new JScrollPane(list);
scroller.setVerticalScrollBarPolicy
    (ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
scroller.setHorizontalScrollBarPolicy
    (ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

panel.add(scroller);
```

Isso é o mesmo que ocorre com JTextArea – você criará um JScrollPane (e o fornecerá à lista) para em seguida adicionar o painel de rolagem (e NÃO a lista) ao painel geral.

- ② **Configure a quantidade de linhas a serem exibidas antes da rolagem.**

```
list.setVisibleRowCount(4);
```
- ③ **Restrinja o usuário à seleção de somente UMA coisa de cada vez.**

```
list.setSelectionMode(ListSelectionMode.
    SINGLE_SELECTION);
```
- ④ **Registre-se para ouvir eventos de seleção na lista.**

```
list.addListSelectionListener(this);
```
- ⑤ **Manipule eventos (descubra o que foi selecionado na lista).**

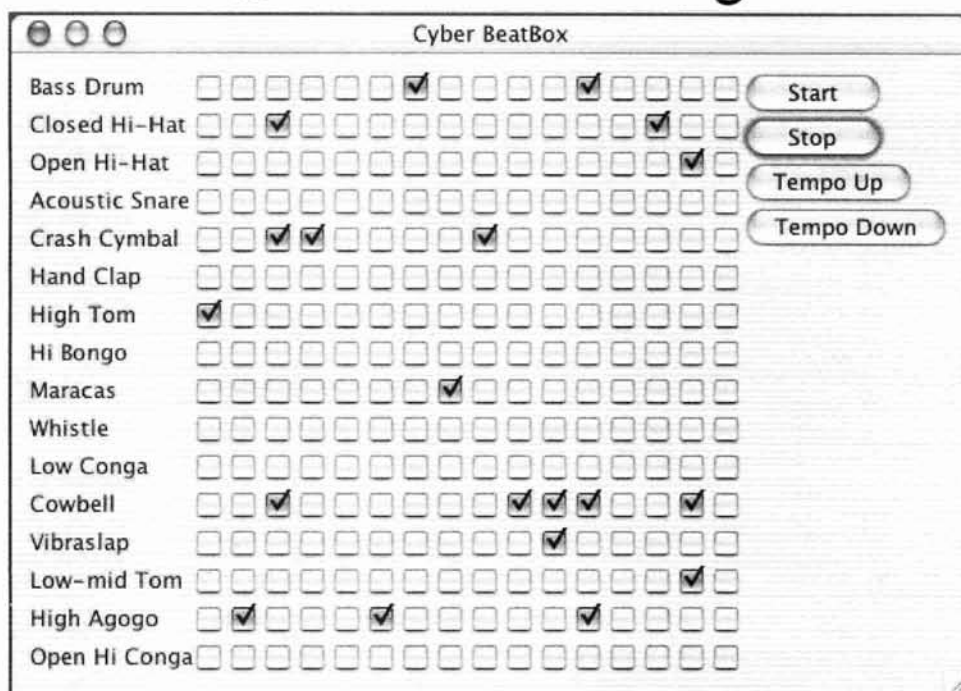
```
public void valueChanged(ListSelectionEvent lse) {
```

Você capturará o evento DUAS VEZES se não inserir esse teste if.

```
    if(!lse.getValueIsAdjusting()) {
        String selection = (String)
            list.getSelectedValue();
        System.out.println(selection);
    }
}
```

Na verdade getSelectedValue() retornará um tipo Object. Uma lista não está limitada somente a objetos String.

Receita de código



Essa parte é opcional. Estamos criando a BeatBox completa, com a GUI e todo o resto. No capítulo Salvando Objetos, aprenderemos como salvar e restaurar padrões de bateria. Para concluir, no capítulo sobre rede (Crie uma Conexão), converteremos a BeatBox em um cliente de bate-papo funcional.

Criando a BeatBox

Essa é a listagem completa do código dessa versão da BeatBox, com botões para a inicialização, a interrupção e a alteração do ritmo. A listagem do código está completa, e totalmente comentada, mas aí vai uma visão geral:

- ① Construa uma GUI com 256 caixas de seleção (JCheckBox) inicialmente desmarcadas, 16 rótulos (JLabel) para os nomes dos instrumentos e quatro botões.
- ② Registre um ActionListener para cada um dos quatro botões. Não precisamos de ouvintes para cada caixa de seleção, porque não estamos tentando alterar o padrão de som dinamicamente (isto é, quando o usuário marcar uma caixa). Em vez disso, esperaremos até que o usuário pressione o botão 'start' e, em seguida, percorreremos todas as 256 caixas de seleção para capturar seu estado e gerar uma faixa MIDI.
- ③ Configure o sistema MIDI (você já fez isso antes) incluindo a captura de um seqüenciador, a criação de uma seqüência e de uma faixa. Estamos usando um método do seqüenciador que é novo na Java 5.0, `setLoopCount()`. Esse método permitirá que você especifique quantas vezes quer que uma seqüência seja

repetida. Também estamos usando o fator de ritmo da sequência para diminuir ou acelerá-la, e manter o novo ritmo de uma iteração do loop até a próxima.

- ④ Será quando o usuário pressionar 'start' que a ação real começará. O método de manipulação de eventos do botão 'start' chamará o método `buildTrackandStart()`. Nesse método, percorreremos todas as 256 caixas de seleção (uma linha de cada vez, todas as 16 batidas de um único instrumento) para capturar seu estado e, em seguida, usaremos as informações para construir uma faixa MIDI [empregando o prático método `makeEvent()` que usamos no capítulo anterior]. Quando a faixa estiver construída, iniciaremos o seqüenciador, que continuará a reprodução (porque ela estará sendo repetida) até o usuário pressionar 'stop'.

```
import java.awt.*;
import javax.swing.*;
import javax.sound.midi.*;
import java.util.*;
import java.awt.event.*;
```

```
public class BeatBox {
```

```
    JPanel mainPanel;
    ArrayList<JCheckBox> checkboxList;
    Sequencer sequencer;
    Sequence sequence;
    Track track;
    JFrame theFrame;
```

Armazenaremos as caixas de seleção em uma `ArrayList`.

Esses são os nomes dos instrumentos, como uma array de `Strings`, para a construção dos rótulos da GUI (em cada linha).

```
String[] instrumentNames = {"Bass Drum", "Closed Hi-Hat", "Open Hi-Hat", "Acoustic Snare", "Crash Cymbal", "Hand Clap", "High Tom", "Hi Bongo", "Maracas", "Whistle", "Low Conga", "Cowbell", "Vibraslap", "Low-mid Tom", "High Agogo", "Open Hi Conga"};
int[] instruments = {35,42,46,38,49,39,50,60,70,72,64,56,58,47,67,63};
```

Esses números representam as 'teclas' reais da bateria. O canal da bateria é como um piano, exceto pelo fato de cada 'tecla' do piano ser um elemento de bateria diferente. Portanto, o número '35' é a tecla do bumbo (bass drum), 42 é o Hi-Chapéu Closed (Closed Hi-Hat), etc.

```
public static void main (String[] args) {
    new BeatBox2().buildGUI();
}
```

```
public void buildGUI() {
    theFrame = new JFrame("Cyber BeatBox");
    theFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    BorderLayout layout = new BorderLayout();
    JPanel background = new JPanel(layout);
    background.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
```

Uma 'borda vazia' nos fornecerá uma margem entre as bordas do painel e onde os componentes estão posicionados. Furamente estético.

```
checkboxList = new ArrayList<JCheckBox>();
Box buttonBox = new Box(BoxLayout.Y_AXIS);

JButton start = new JButton("Start");
start.addActionListener(new MyStartListener());
buttonBox.add(start);
```

```
JButton stop = new JButton("Stop");
stop.addActionListener(new MyStopListener());
buttonBox.add(stop);
```

Nada de especial aqui, apenas código da GUI. Grande parte você já viu.

```
JButton upTempo = new JButton("Tempo Up");
upTempo.addActionListener(new MyUpTempoListener());
buttonBox.add(upTempo);

JButton downTempo = new JButton("Tempo Down");

downTempo.addActionListener(new MyDownTempoListener());
buttonBox.add(downTempo);

Box nameBox = new Box(BoxLayout.Y_AXIS);
for (int i = 0; i < 16; i++) {
    nameBox.add(new Label(instrumentNames[i]));
}
```

```

background.add(BorderLayout.EAST, buttonBox);
background.add(BorderLayout.WEST, nameBox);

theFrame.getContentPane().add(background);

GridLayout grid = new GridLayout(16,16);
grid.setVgap(1);
grid.setHgap(2);
mainPanel = new JPanel(grid);
background.add(BorderLayout.CENTER, mainPanel);

for (int i = 0; i < 256; i++) {
    JCheckBox c = new JCheckBox();
    c.setSelected(false);
    checkboxList.add(c);
    mainPanel.add(c);
} // fim do loop

setUpMidi();

theFrame.setBounds(50,50,300,300);
theFrame.pack();
theFrame.setVisible(true);
} // fecha o método

```

Ainda é código de configuração da GUI. Nada de especial.

```

public void setUpMidi() {
    try {
        sequencer = MidiSystem.getSequencer();
        sequencer.open();
        sequence = new Sequence(Sequence.PPQ,4);
        track = sequence.createTrack();
        sequencer.setTempoInBPM(120);

    } catch (Exception e) {e.printStackTrace();}
} // fecha o método

```

Cria as caixas de seleção, configura-as com 'false' (para que não estejam marcadas) e as adiciona à ArrayList E ao painel da GUI.

O costumieiro trecho de configuração MIDI para a captura do seqüenciador, da seqüência e da faixa. Novamente, nada de especial.

É aqui que tudo acontece! É o local em que converteremos o estado da caixa de seleção em eventos MIDI e os adicionamos à faixa.

```

public void buildTrackAndStart() {
    int[] trackList = null;

    sequence.deleteTrack(track);
    track = sequence.createTrack();

    for (int i = 0; i < 16; i++) {

        int key = instruments[i];

```

Criaremos uma matriz de 16 elementos para armazenar os valores de um instrumento, com todas as 16 batidas. Se o instrumento tiver que ser reproduzido nessa batida, o valor desse elemento será a tecla. Se esse instrumento NÃO tiver que ser reproduzido nessa batida, insira um zero.

Elimina a faixa antiga, cria uma nova.

Fará isso para cada uma das 16 LINHAS (isto é, Bass, Congo, etc.).

Configura a 'tecla' que representará qual é esse instrumento (bumbo, hi-chapéu, etc. A matriz de instrumentos contém os números MIDI reais de cada instrumento).

```

    for (int j = 0; j < 16; j++) {

```

Fará isso para cada uma das BATIDAS dessa linha.

```

        JCheckBox jc = (JCheckBox) checkboxList.get(j + (16*i));
        if ( jc.isSelected()) {
            trackList[j] = key;
        } else {
            trackList[j] = 0;
        }
    } // fecha o loop interno

```

A caixa de seleção dessa batida está selecionada? Se estiver, insira o valor da tecla nessa posição da matriz (a posição que representa essa batida). Caso contrário, o instrumento NÃO deve reproduzir essa batida, portanto, configure-a com zero.

```
makeTracks(trackList);
track.add(makeEvent(176,1,127,0,16));
} // fecha o loop externo
```

Para esse instrumento, e para todas as 16 batidas, cria eventos e os adiciona à faixa.

```
track.add(makeEvent(192,9,1,0,15));
try {
```

Queremos nos certificar sempre de que HÁ um evento na batida 16 (ela vai de 0 a 15). Caso contrário, a BeatBox pode não percorrer todas as 16 batidas antes de começar novamente.

```
sequencer.setSequence(sequence);
sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
sequencer.start();
sequencer.setTempoInBPM(120);
} catch(Exception e) {e.printStackTrace();}
} // fecha o método buildTrackAndStart
```

Permite que você especifique a quantidade de iterações do loop ou, nesse caso, um loop contínuo.

AGORA REPRODUZA!!

```
public class MyStartListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        buildTrackAndStart();
    }
} // fecha a classe interna
```

A primeira das classes internas são os ouvintes dos botões. Nada de especial aqui.

```
public class MyStopListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        sequencer.stop();
    }
} // fecha a classe interna
```

```
public class MyUpTempoListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * 1.03));
    }
} // fecha a classe interna
```

Os ouvintes das outras classes internas dos botões

TempoFactor dimensionará o ritmo da sequência pelo fator fornecido. O padrão é 1,0, portanto, estamos ajustando para cerca de 3% por clique.

```
public class MyDownTempoListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * .97));
    }
} // fecha a classe interna
```

Isso criará eventos para um instrumento de cada vez, para todas as 16 batidas. Portanto, pode capturar um int[] para o bumbo e cada índice da matriz conterá a tecla desse instrumento ou um zero. Se tiver um zero, o instrumento não deve ser reproduzido nessa batida. Caso contrário, um evento será criado e adicionado à faixa.

```
public void makeTracks(int[] list) {
    for (int i = 0; i < 16; i++) {
        int key = list[i];

        if (key != 0) {
            track.add(makeEvent(144,9,key, 100, i));
            track.add(makeEvent(128,9,key, 100, i+1));
        }
    }
}
```

Cria os eventos NOTE ON e NOTE OFF e os adiciona à faixa.

```
public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
    MidiEvent event = null;
    try {
        ShortMessage a = new ShortMessage();
        a.setMessage(comd, chan, one, two);
        event = new MidiEvent(a, tick);

    } catch(Exception e) {e.printStackTrace();}
    return event;
}
```

Esse é o método utilitário da receita de código do último capítulo. Nada de novo.

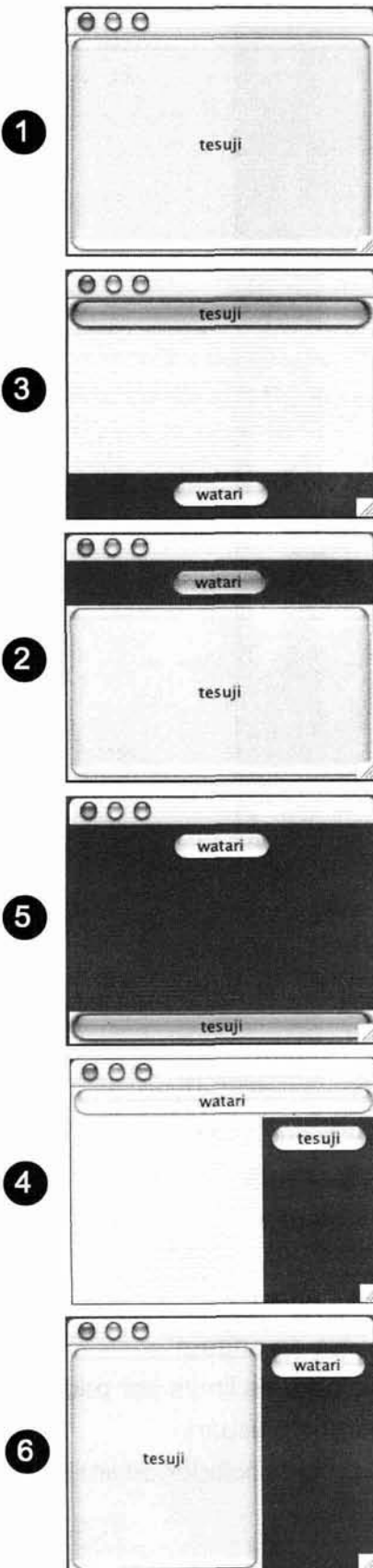
} // fecha a classe



Que código está relacionado a que layout?

Cinco das seis telas abaixo foram criadas a partir de um dos trechos de código ao lado. Ligue cada um dos cinco trechos de código ao layout que ele produziria.

Exercício



Trechos de código

D

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.NORTH, panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);
```

B

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

C

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);
```

A

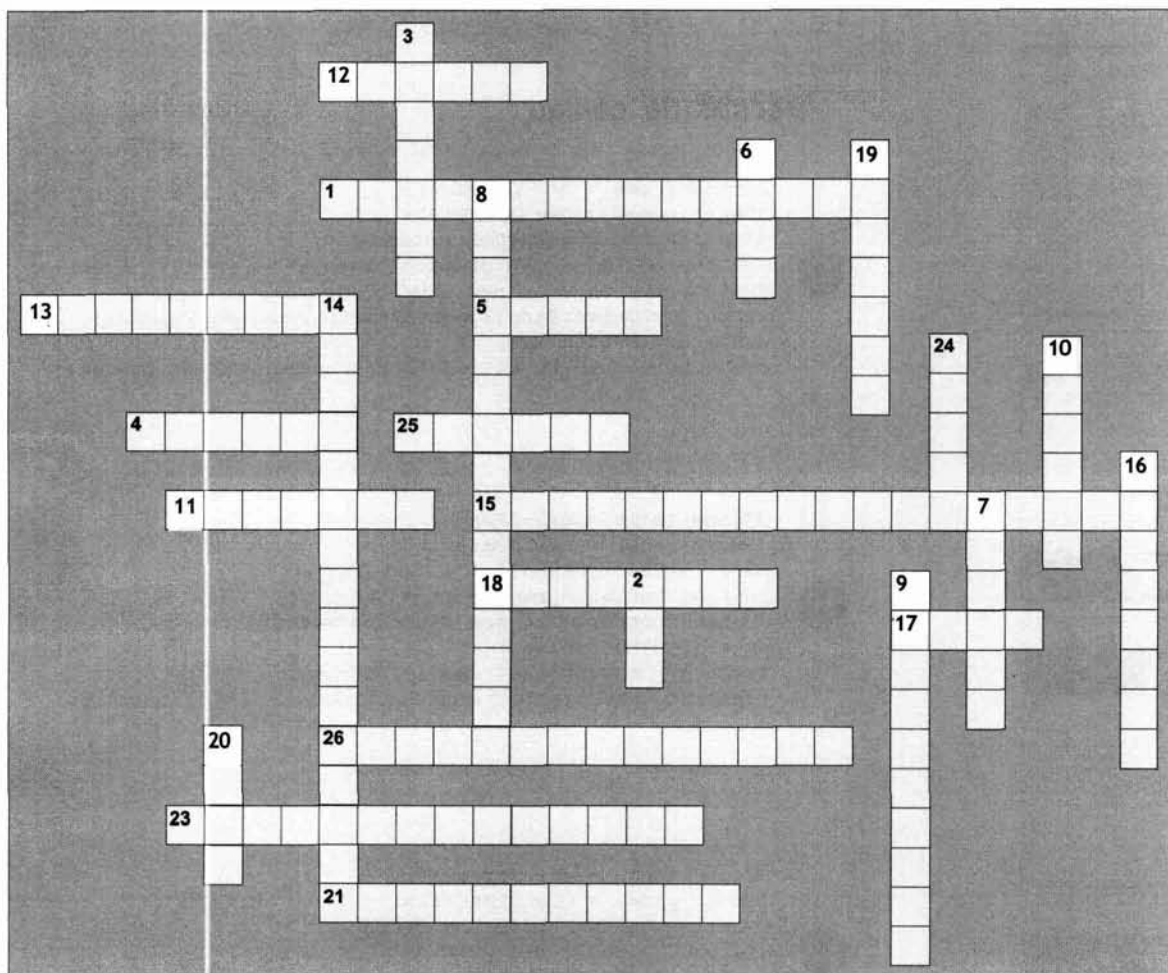
```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(button);
frame.getContentPane().add(BorderLayout.NORTH, buttonTwo);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

E

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.SOUTH, panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.NORTH, button);
```



Cruzadas GUI 7.0



Você consegue.

Horizontais

1. O playground do artista
4. Local para o que restar do gerenciador de limite
5. Aparência Java
11. Objeto genérico de espera
12. Um acontecimento
13. Aplicar um elemento gráfico
15. Padrão de JPanel
17. Teste polimórfico
18. Mova-se
21. Muito a dizer
23. Selecione várias
25. Companheiro do botão
26. Casa de actionPerformed

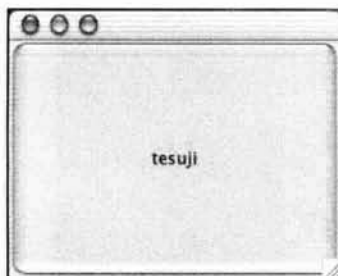
Verticais

2. Pai do Swing
3. Jurisdição da moldura
6. Casa da ajuda
7. Mais diversão do que texto
8. Gíria para componente
9. Comando de Romulin
10. Disposição
16. Regras do gerenciador
14. Comportamento da origem
19. Usa o gerenciador de limite por padrão
20. Comportamento do usuário
24. Lado direito do gerenciador de limite



Solução dos Exercícios

1



C

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
```

2



D

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.NORTH,panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
```

3



E

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.SOUTH,panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.NORTH,button);
```

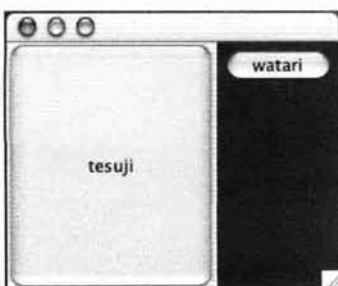
4



A

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(button);
frame.getContentPane().add(BorderLayout.NORTH,buttonTwo);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

6



B

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
frame.getContentPane().add(BorderLayout.EAST, panel);
```



Respostas do Quebra-Cabeças

Cruzadas GUI 7.0

