JAVA 5

Interface gráfica

ESTE MÓDULO APRESENTA OS CONCEITOS POR TRÁS DA PROGRAMAÇÃO DA INTERFACE GRÁFICA DO USUÁRIO (GUI) e apresenta a API Abstract Window Toolkit (AWT) – um conjunto de ferramentas independente de plataforma para o desenvolvimento de aplicações gráficas. É demonstrado o uso de gerentes de layout, da biblioteca de componentes e do tratamento de eventos.

Tópicos abordados neste módulo

Objetivos

No final deste módulo você deverá ser capaz de:

1.1. Programação da AWT

Java oferece uma biblioteca de objetos independentes de plataforma para a programação da Interface Gráfica (GUI). Esses objetos compõem o *Abstract Window Toolkit (AWT)*, que é um conjunto de ferramentas (classes, interfaces e métodos) abstratas para o desenvolvimento de aplicações que utilizam o sistema de janelas.

Hoje, com a grande base instalada de computadores com monitores gráficos de alta resolução, nenhum software que dependa de uma interface orientada a caractere consegue competir no mercado. O AWT oferece botões, caixas de texto, janelas, quadros, barras de rolamento, e todos os objetos e mecanismos

necessários para desenvolver aplicações gráficas, orientadas a eventos, em qualquer das plataformas que suportam o ambiente de execução (*runtime*) Java.

O nível de abstração do AWT é alto o suficiente para garantir a independência de plataforma. O AWT não determina a forma de apresentação das janelas; tarefa que fica com o sistema operacional nativo. Desta forma, um programa gráfico rodando no ambiente X-Window/ Motif usa os botões, barras de rolamento e caixas de rádio característicos da interface Motif, enquanto o mesmo programa rodando no Macintosh ou Windows95 irá se apresentar com os objetos de janelas característicos de seus respectivos sistemas. Um outro pacote para desenvolvimento de aplicações gráficas – o JFC (*Java Foundation Classes*) ou *Swing*, pode ser usado para desenvolver interfaces gráficas mais sofisticadas, independentes dos recursos oferecidos por cada sistema operacional. Aplicações Swing podem ou não ter o *look&feel* de aplicações nativas.

Neste capítulo, estudaremos as classes mais importantes do pacote `java.awt`.

Lógica da Programação Orientada a Eventos

A principal diferença entre os programas que rodam em ambientes de janelas e os programas que executam na linha de comando está na forma como são executados. A lógica de um é o inverso da outra. Em programas de linha de comando as ações ocorrem seqüencialmente do começo ao fim, podendo-se prever quando e onde o usuário terá opções de entrar com dados ou interferir na execução do programa.

Já um programa orientado a eventos tem uma lógica assíncrona. Não é possível prever qual botão o usuário irá apertar e quando. Em vez de um fluxo contínuo de controle, o programa fica esperando por uma ação do usuário. Para isso, ele tem que ficar monitorando os eventos que ocorrem no sistema operacional, como movimento do mouse e teclas digitadas, para saber se ele precisa realizar alguma ação baseado neles.

Quando o usuário realiza alguma ação (movendo o mouse sobre uma janela, por exemplo), o sistema de janelas captura o evento e o passa para uma rotina que foi criada para manuseá-lo. A rotina deverá lidar com o evento e com a ação que está associada a ele. Por exemplo, se um mouse é clicado sobre o botão ☒ no canto superior direito de um programa Windows95, a rotina deve tomar

conhecimento que esse evento ocorreu e providenciar a execução de uma resposta ao evento: geralmente o fechamento da janela ou a chamada de uma rotina mais complexa para lançar uma janela de diálogo para confirmar a saída.

Em Java, todos os eventos do sistema operacional podem ser capturados e manuseados através das subclasses de `java.awt.AWTEvent`. O tratamento de eventos é realizado por um objeto especialmente criado para este fim. Este objeto deve implementar os métodos de manuseio de eventos para o tipo de evento desejado. A classe que produz os eventos cadastra objetos que desejam ser informados sobre o evento que irá acontecer. O exemplo abaixo mostra uma aplicação onde o aperto de um botão provoca o fechamento do programa.

```
// classe que trata evento de fecha janela
class WindowCloser extends WindowAdapter {
   public void windowClosing(WindowEvent e) {
      System.exit(0);
   }
}  // end class


// classe (janela) onde que produz o evento
(...)
this.addWindowListener(this.new WindowCloser());
(...)
```

Quando o botão de fechar janela é apertado, um objeto *WindowCloser* é criado e seu método `windowClosing()` é executado, encerrando o programa. Outras respostas poderiam ocorrer ao mesmo evento, bastava cadastrar novas classes manuseadores de eventos usando `addWindowListener()`. Eventos de ação (apertar botão, selecionar menu) são sempre tratados como `Action` events. Outros tipos de eventos são `ItemEvent`, `MouseEvent`, etc. Cada qual tem seus próprios ouvintes (*listeners*) e métodos.

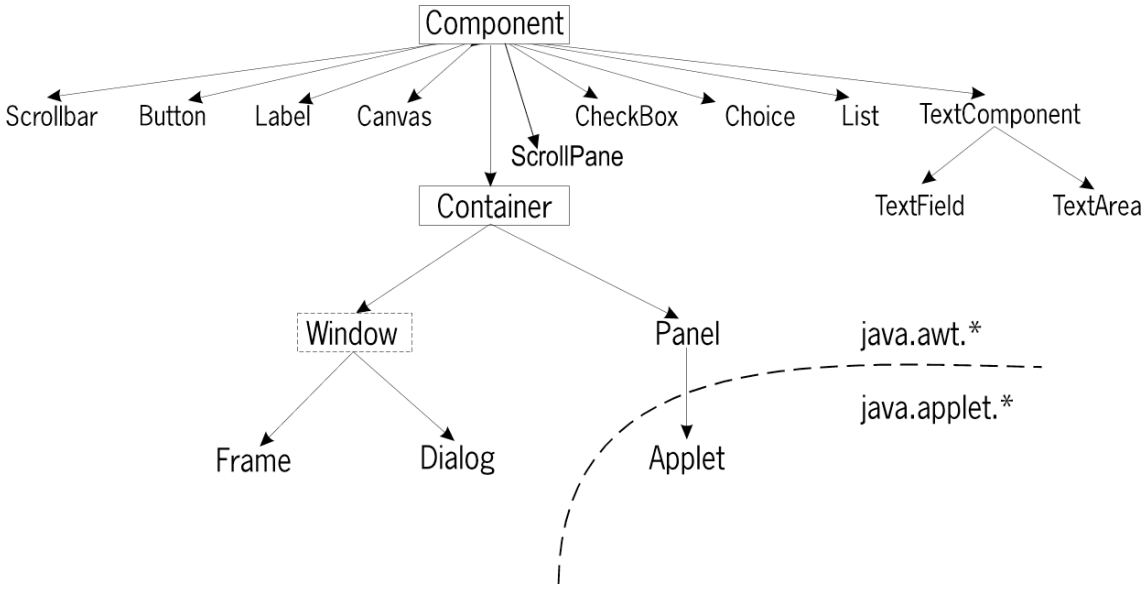
Hierarquia e Funções do AWT

O pacote `java.awt`, que é o *Abstract Window Toolkit*, pode ser dividido em quatro categorias:

- **Subclasses de Component e MenuComponent:** um conjunto de objetos da GUI: "widgets" (menus, botões, caixas de escolha, caixas de rádio, etc.) e recipientes (janelas, quadros, painéis, caixas de diálogo).

- **Sistema de manuseio de eventos da AWT:** classe `AWTEvent` e suas subclasses (`pacote java.awt.event`) e métodos de captura de eventos.
- **Gerentes de layout:** organizam os "widgets" em um "container" de acordo com uma estrutura definida. (implementações da interface `LayoutManager`).
- **Classes de suporte a operações gráficas:** diversas classes que permitem desenhar um polígono, preencher uma elipse, mudar uma cor, mostrar uma imagem, etc.

A figura abaixo ilustra a hierarquia dos componentes da GUI que são subclasses de `Component` ("widgets" e "containers").



Componentes e Recipientes (Containers)

Como ilustra a figura acima, `Containers` são uma sub-classe dos `Components`, assim como todos os widgets. Os widgets, que são componentes, podem ser colocados em recipientes (`containers`), que também são componentes e por sua vez podem ser colocados em outros recipientes. A diferença entre os dois é ilustrado abaixo:

Components	Containers (são Components)
1. São exibidos na tela mostrando todo o seu	1. Recebem e organizam widgets de acordo

conteúdo.	com um controle de layout.
2. Capturam e lidam com eventos relacionados aos widgets, teclado e mouse.	2. Sobrepõem métodos de um componente, para captura e manuseio de eventos.

Component

É a superclasse de todos os objetos da GUI (exceto menus). Têm uma posição (x, y) e uma dimensão (altura, largura), e podem ser desenhados na tela. Não é possível instanciar um objeto `Component` diretamente (ele não tem método construtor), somente uma das suas subclasses. `Component` define vários métodos, todos herdados pelas suas subclasses. O método mais importante de `Component` é:

```
public void paint(Graphics g)
```

Este método, quando chamado pelo sistema de tempo de execução, desenha o componente e todo o seu conteúdo na tela.

Além deste método, `Component` ainda possui dezenas de outros que alteram sua posição e tamanho (`setBounds`, `setSize`), e alteram propriedades gráficas.

Container

Implementa um componente que pode conter outros componentes e tratá-los como um grupo. Também não pode ser instanciado diretamente. Você deve usar uma das suas subclasses `Panel`, `Frame` ou `Dialog` (subclasses de `Window`). O principal método de `Container`, herdado por todas as suas subclasses, é:

```
public Component add(Component c)
public Component add("North", Component c)
```

que adiciona um componente ao `Container`. A maioria dos outros métodos são relacionados com gerentes de layout.

Window

É uma janela em branco, sem bordas ou barra de menu. É um recipiente cujo layout default é `BorderLayout` (veja adiante). Pode ser instanciada, mas em geral não é usada diretamente. Utiliza-se muito as suas subclasses `Frame` e `Dialog`.

Frame

É uma especialização de `Window`, que tem uma borda, pode ter um título, uma barra de menu, um ícone e um cursor. Várias constantes definidas nesta classe

definem diversos tipos de cursor. `Frame` geralmente é a base de uma aplicação GUI. É o recipiente que recebe todos os componentes que formam a interface gráfica do programa.

Há duas formas de se criar um objeto `Frame`:

```
Frame f = new Frame()
Frame f = new Frame("Título da Janela");
```

A primeira forma cria um quadro sem título. A segunda cria um quadro com o título "Título da Janela" na sua barra de título. Também, pode-se usar:

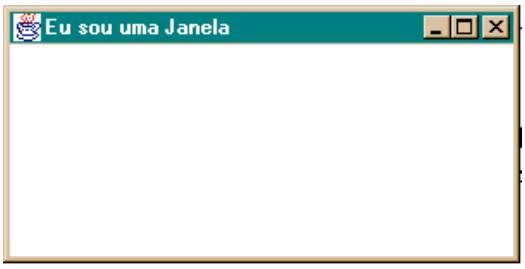
```
f.setTitle("Título da Janela");
```

para definir o título. Outros métodos são: `setCursor(int tipo_cursor)`, que especifica um cursor; `setMenuBar(MenuBar mb)`, que define uma barra de menu e `setIconImage(Image img)`, que define uma imagem para o ícone do quadro.

Uma maneira mais comum de se criar um `Frame` é fazê-lo a base de uma aplicação. **Exemplo:**

```
import java.awt.*;
public class Quadro extends Frame {

    Quadro() {
       super ("Eu sou uma Janela");
       setSize(300,150);
       setLocation(300,200);
       setVisible(true);
    }

    public static void main(String a[]) {
       new Quadro();
    }
}
```

Faz aparecer na tela a janela a seguir.

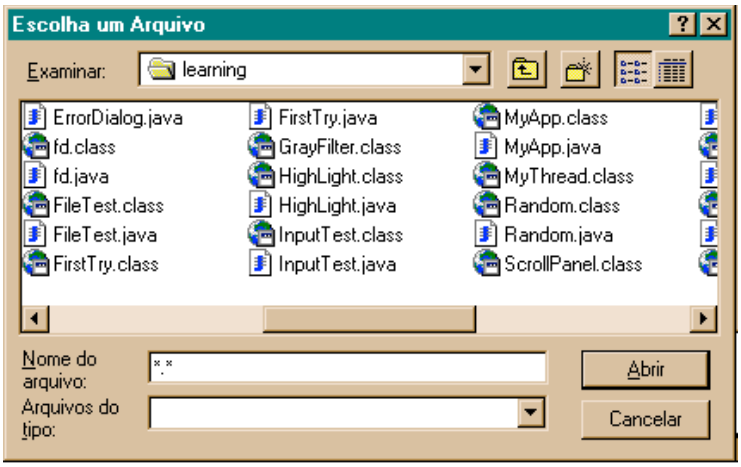


Dialog

É uma janela de caixa de diálogo. Aplicações são diálogos Sim/Não, OK/Cancela, informações, avisos, erros, perguntas, etc. Uma subclasse

importante é `FileDialog`, que faz aparecer a caixa de diálogo para seleção de um arquivo. Adicionando as linhas a seguir no `main` do programa anterior, faz aparecer uma janela `FileDialog`:

```
Quadro() {
    setBounds(300,150, 300,200);
    FileDialog fd = new FileDialog(this, "Escolha" +
                                   " um Arquivo");
    fd.setVisible(true);
    add(fd);
    setVisible(true);
}
```



Panel

É um recipiente (`Container`) que não cria uma janela própria (como faz `Dialog` e `Frame`). É útil para dividir partes de uma interface maior (como um `Frame` ou `Dialog`) ou mesmo outro `Panel`.

`Panel` quase não tem métodos próprios. Sua função básica é proporcionar um layout para a sua sub-classe `Applet` e como forma de dividir um layout complexo.

Applet

Applet é uma subclasse de `Panel` que faz parte do pacote `java.applet`. Applets são discutidos no final deste capítulo.

1.2. Eventos da AWT

Em Java, todo evento é uma subclasse de `java.util.EventObject`. Os eventos do AWT são subclasses de `java.awt.AWTEvent` e os diversos tipos básicos pré-definidos de eventos do AWT como `MouseEvent`, `WindowEvent`, `ActionEvent`, etc. são parte do novo `java.awt.event`.

Todo evento tem um objeto origem que é obtido através do seu método `getSource()`. Cada evento do AWT tem um tipo que é obtido com `getID()`. A identificação é usada para distinguir entre os diversos tipos de eventos que ocorrem em uma mesma classe (como `MouseEvent.MOUSE_CLICKED` e `MouseEvent.MOUSE_EXITED`). As classes contém os métodos que têm algo a ver com o tipo de evento ocorrido. Por exemplo, `MouseEvent` tem métodos `getX()`, `getY()`, etc.

O modelo de eventos baseia-se no conceito de um objeto de escuta (*listener*). Qualquer objeto interessado em receber eventos é um *listener*. Um objeto que gera eventos (*event source*) mantém uma lista de *listeners* interessados em serem notificados quando um determinado evento ocorrer. Eles também possuem métodos para que os *listeners* se cadastrem para receber eventos.

A notificação do *listener* pela fonte ocorre através de uma invocação de método do *listener*, passando para ele uma instância de `EventObject`. É necessário que todos os *listeners* implementem o método necessário para que possam ser avisados. Isto é feito através da implementação de uma ou mais interfaces apropriadas para o tipo de evento que se deseja receber. Os métodos implementados recebem um único argumento que é o objeto (instância de `EventObject`) que corresponde ao *listener*. O objeto deve conter toda a informação necessária para que um programa possa responder ao evento.

A tabela abaixo relaciona tipos de evento, métodos e interfaces de escuta (*listeners*) [Java In a NutShell. 2nd. Ed.]:

Classe	**Interface (listener)**	**Métodos (listener)**
`ActionEvent`	`ActionListener`	`actionPerformed()`
`AdjustmentEvent`	`AdjustmentListener`	`adjustmentValueChanged()`

`ComponentEvent`	`ComponentListener`	`componentHidden()`
		`componentMoved()`
		`componentResized()`
		`componentShown()`

`ContainerEvent`	`ContainerListener`	`componentAdded()`

		componentRemoved()
FocusEvent	FocusListener	focusGained() focusLost()
ItemEvent	ItemListener	itemStateChanged()
KeyEvent	KeyListener	keyPressed() keyReleased() keyTyped()
MouseEvent	MouseListener	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()
	MouseMotionListener	mouseDragged() mouseMoved()
TextEvent	TextListener	textValueChanged()
WindowEvent	WindowListener	windowActivated() windowClosed() windowClosing() windowDeactivated() windowDeiconified() windowIconified() windowOpened()

Para ter acesso aos eventos e seus métodos, deve-se importar as classes de `java.awt.event.*`. Todos os métodos de eventos sempre têm a forma:

```
public void nome (XXXEvent evt) { ... }
```

onde `XXX` é o tipo de evento (coluna 1) tratado. Um evento só será encaminhado para um ouvinte se ele estiver cadastrado na fonte através de um método do tipo:

```
fonte.addXXXListener(ouvinte do tipo XXXListener);
```

onde fonte é o objeto-fonte onde ocorre o evento e `XXX` o tipo do evento (`Mouse`, `Action`, `Window`, etc.). O ouvinte do tipo `listener` é um objeto criado com a classe que implementa a interface correspondente ao tipo de evento tratado.
Por exemplo, para cadastrar um botão para que ele responda a eventos de clique do mouse sobre ele (eventos do tipo `Action`) é preciso fazer:

```
Button b = new Button("Botão"); //criação
b.addActionListener(new OuvinteAcao());
```

considerando-se que existe uma classe chamada `OuvinteAcao()` que implementa a interface `ActionListener` (e consequentemente o método `actionPerfor-`

`med()`). O método deverá descrever as ações a serem tomadas quando o evento ocorrer.

1.3. Componentes da AWT

Os componentes são geralmente são adicionados aos `Frames`, `Dialogs` ou `Panels`. Para criar um componente, é preciso declarar uma referência para ele e instanciá-lo. Em seguida, deve ser adicionado a um recipiente. Quando o recipiente se tornar visível, todos os componentes que ele tiver aparecerão na tela. O posicionamento e dimensionamento do componente geralmente é ditado pela política de *layout* do recipiente. Caso o recipiente não tenha política de *layout* (pode ter sido desligado com `setLayout(null)`), é preciso ainda definir as dimensões, margens e localização do componente.

Os passos para a criação e utilização dos componente são:

1. Declarar a referência (geralmente como membro do objeto que constrói a interface gráfica):

   ```
   private Button b1;
   ```

2. Instanciar o objeto.

   ```
   b1 = new Button("Enviar Dados");
   ```

3. Adicioná-lo ao recipiente (ou a um painel dentro do recipiente). A forma de colocar um componente dentro de outro depende do *layout* utilizado. Para o `FlowLayout`, poderia ser:

   ```
   this.add(b1);
   ```

4. Cadastrar o objeto com um manuseador (ouvinte) de eventos

   ```
   b1.addActionListener(new MenusEvt);
   ```

5. Definir um comando para identificar o evento

   ```
   b1.setActionCommand("env");
   ```

Para os itens 4 e 5 acima, estamos supondo a existência de uma classe `MenusEvt` que poderia ter a seguinte estrutura:

```
class MenusEvt implemens java.awt.event.ActionListener {
  public void actionPerformed(ActionEvent e) {
    String comando = e.getActionCommand();
    ...
```



```
    }
}
```

Scrollbar

Barra de rolamento. Exemplo de criação:

```
Scrollbar s =
    new Scrollbar(Scrollbar.VERTICAL, 50, 10, 0, 99);
```

Os argumentos são respectivamente: a orientação da barra (`Scrollbar.HORIZONTAL` ou `Scrollbar.VERTICAL`), a posição inicial da barra deslizante, o tamanho da parte visível da área e os valores do início e fim da escala. Há vários métodos para controlar a posição das barras. Para capturar o valor da posição selecionada, deve-se criar um manuseador de eventos que implemente a interface `AdjustmentListener` (evento `AdjustmentEvent`) e cadastrar o `Scrollbar` com `addAdjustmentListener()`.

Button

Botão. Exemplo de criação:

```
Button b1 = new Button("Aperte-me");
```

Para capturar o evento, implemente um `ActionListener`. Para cadastrá-lo, use `addActionListener()`. Veja exemplo na seção anterior.

Canvas

É um componente que não cria um desenho *default* ou qualquer mecanismo próprio de manuseio de eventos. É uma tela que serve para desenhar gráficos ou receber entrada do usuário. `Canvas` normalmente é utilizada através de uma subclasse para sobrepor o seu método `paint` e obter um contexto gráfico onde se possa usar os métodos de `Graphics` (para desenhar retângulos, elipses, polígonos, imagens, etc.).

O exemplo abaixo cria um objeto `Canvas` e o adiciona ao contexto atual:

```
Canvas cv = new Canvas();
cv.setBackground(Color.lightGray);
cv.resize(70, 30);
this.add(cv);
```

Se for necessário redirecionar o teclado para um `Canvas`, é preciso requisitar o foco do mesmo para este componente usando o seu método

`requestFocus()`. Para desenhar, deve-se usar pode-se cadastrar o `Canvas` como fonte de eventos de mouse (`MouseEvent`)

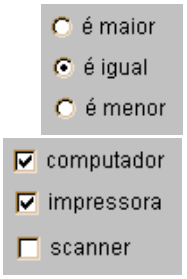
Label

Rótulo de texto flutuante somente-leitura. Usado para rotular outros objetos de um recipiente. Possui métodos para fazer o alinhamento com outros objetos. Exemplo:

```
this.add(new Label("List of Widgets"));
```

Você pode fazer um `Label` responder a eventos da mesma forma que com `Canvas` tratando eventos do teclado com `KeyEvent` e usando `requestFocus()`, e usando eventos do mouse para realizar outras operações.

CheckBox



Caixa de seleção. Possui um método `boolean` que indica se está selecionada ou não (ligada ou desligada). Opcionalmente, pode fazer parte de um `CheckboxGroup`. Apenas um dos objetos `Checkbox` de um `CheckboxGroup` pode ser selecionado ao mesmo tempo.

As três caixas de seleção a seguir, pertencem a o grupo `g` e são exibidas como caixas de rádio. Somente uma das três pode ser selecionada. A segunda está inicialmente selecionada.

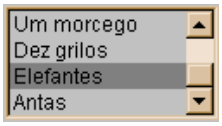
```
CheckboxGroup g = new CheckboxGroup();
this.add(new Checkbox("é maior", g, false));
this.add(new Checkbox("é igual", g, true));
this.add(new Checkbox("é menor", g, false));
```

As caixas de seleção abaixo não pertencem a um grupo. Pode-se selecionar mais de uma.

```
this.add(new Checkbox("computador"));
this.add(new Checkbox("impressora"));
this.add(new Checkbox("scanner"));
```

O evento apenas informa se o estado foi mudado ou não (não diz qual é o estado). O tipo de evento é `ItemEvent`. O manuseador pode ser implementado com `ItemListener` e cadastrado com `addItemListener`.

List



Lista de opções rolante, da qual o usuário pode selecionar uma ou mais opções. Exemplo:

```
List li = new List(4, false);
// false indica que lista não aceita múltiplas seleções
// 4 é o número de opções que será mostrada

li.reshape(0, 0, 100, 100);   // redimensiona
li.add("Um sapo");
li.add("Uma barata");
li.add("Dois ratos");
li.add("Um caipira");
li.add("Um morcego");
li.add("Dez grilos");
li.add("Elefantes");
li.add("Antas");
li.select(3);    // item previamente selecionado é o 4°

this.add(li);
```

Para receber a opção do usuário, implementa-se um `ItemListener`, como em `Checkbox`.

Choice

Lista de opções tipo menu "drop-down". Quando você move o mouse sobre a opção exibida, todas as outras aparecem e você poderá selecionar uma outra opção. Exemplo:

```
Choice opt = new Choice();

opt.add("Daschhund");
opt.add("Dobermann");
opt.add("Collie");
opt.add("Poodle");
opt.add("Vira-Lata");

this.add(opt);
```

Para capturar o evento e obter a seleção do usuário usa-se o mesmo formato utilizado para listas ou `ActionEvent` (para obter a seleção atual).

TextField

Uma linha de texto editável. A maioria dos seus métodos são definidos pela superclasse `TextComponent`, que também são herdados por

`TextArea`, descrito a seguir. Para adicionar um `TextField` no contexto atual, pode-se fazer:

```
TextField tf1 = new TextField("Escreva!"));
TextField tf2 = new TextField(20);
```

`TextField` causa um evento de texto (`TextEvent`) quando o usuário aperta a tecla **ENTER** (ou **RETURN**). Para obter o texto digitado implemente a interface `TextListener` e cadastre o objeto com `addTextListener`. Os métodos de `TextComponent getText()` e `setText(String texto)` servem para obter ou alterar o texto contido no campo de texto.

TextArea

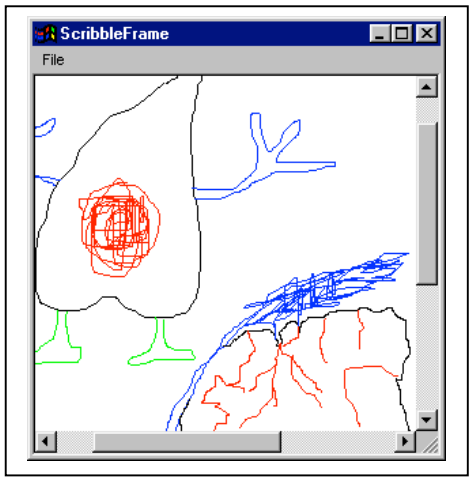
Área de texto editável com várias linhas. É definida informando o número de linhas e colunas e/ou um texto inicial. O exemplo a seguir cria um objeto `TextArea` com 5 linhas visíveis e 20 caracteres de largura:

```
TextArea ta1 = new TextArea(4, 20);
ta1.setText("Você pode digitar aqui!");
```

ou

```
TextArea ta2 = new TextArea("Você pode digitar aqui!",
                                                4, 20);
```

O texto digitado em `TextArea` não gera evento algum, mas fica a disposição para ser lido quando outro evento ocorrer. Por exemplo, pode-se criar um botão que quando apertado, causa um evento e faz com que o conteúdo de `TextArea` seja lido na rotina de manuseio desse evento. Os métodos para ler e escrever texto são os mesmos de `TextComponent` e `TextField`.



ScrollPane

O recipiente `ScrollPane` pode acomodar um único componente filho que pode ser maior que ele mesmo. Na tela aparece uma janela de tamanho fixo que permite a visualização de uma área do componente. Barras de rolamento verticais e/ou horizontais aparecem para permitir a visualização de toda a área.

Para criar um `ScrollPane`, basta instanciá-lo e adicionar um objeto a ele. `ScrollFrame` só suporta um componente e não pode ter um gerente de layouts especificado. Exemplo de uso:

```
ScrollPane pane = new ScrollPane();
pane.setSize(300, 300);
add(pane, "Center");
Panel panel = new Panel ()
panel.resize(500, 500); // Panel é maior que ScrollPane
pane.add(panel);
```

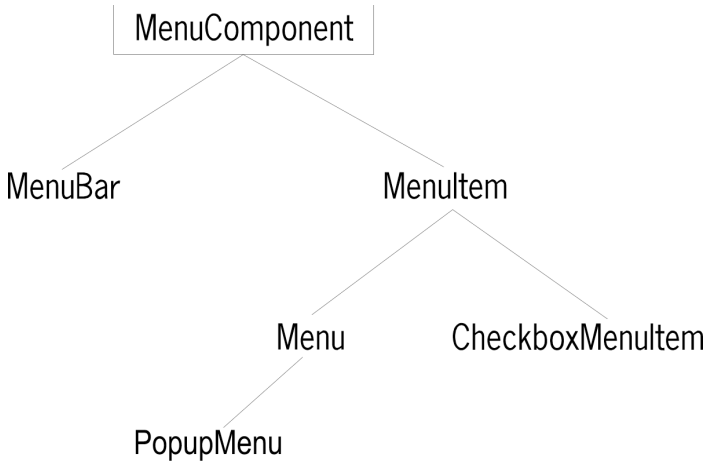
Exercício

1. Desenvolva uma aplicação em Java que consista de um `Frame` e pelo menos um de cada um dos objetos acima. Use um `Panel p` dentro do frame e adicione cada componente no painel usando `p.add(componente)`. Para adicionar o `Panel`, use no construtor:

```
Panel p;
this.add (BorderLayout.CENTER, p);
```

Menus

Menus em Java não são descendentes da classe `Component` mas da uma classe específica, chamada de `MenuComponent`. Através da interface `MenuContainer`, podem ser incluídos apenas em `Frames` (a única exceção é `PopupMenu`). A única forma de incluir menus comuns em uma applet, é através da criação de um objeto `Frame`, separado da applet. A figura abaixo ilustra a hierarquia dos objetos de menu. O `PopupMenu` é o único, da família de menus que pode ser incluído em applets.



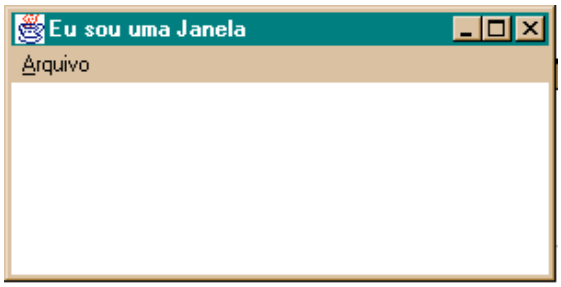
MenuComponent

`MenuComponent`, assim como `Component`, é uma classe raiz da AWT (é uma especialização de `Object`). A classe `MenuComponent` é a superclasse de todas as classes relacionadas com menus. Ela raramente é usada diretamente e seus métodos são utilizados através de suas subclasses `MenuBar` e `MenuItem`.

MenuBar

Cria uma barra de menu que pode ser incluída em um objeto `Frame` (através do método `setMenuBar()`). Menus podem ser adicionados usando o seu método `add()` e `setHelpMenu()`. O exemplo a seguir mostra uma barra de menu dentro de um objeto Frame:

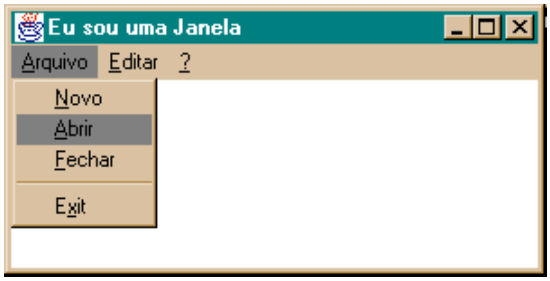
```
import java.awt.*;
public class Quadro {
    static Frame f = new Frame ("Eu sou uma Janela");

    public static void main(String a[]) {
      f.setBounds(300,150,300,200);
      MenuBar mb = new MenuBar(); // cria barra
      mb.add(new Menu("Arquivo")); // adiciona menu
      f.setMenuBar(mb);       // coloca em frame
      f.setVisible(true);
    }
}
```

Menu

Cria um painel tipo "pull-down" que aparece no `MenuBar` (como o menu Arquivo no exemplo acima). Cada `Menu` pode conter várias opções que são objetos do tipo `MenuItem`. No exemplo a seguir, acrescentamos mais dois menus e algumas opções para o primeiro deles:

```
(...)
MenuBar mb = new MenuBar();
  Menu m1 = new Menu("Arquivo"); // Novo menu 1
    m1.add(new MenuItem("Novo"));  // Criação e
    m1.add(new MenuItem("Abrir")); // adição das
    m1.add(new MenuItem("Fechar"));// opções do
    m1.addSeparator();              // menu 1
    m1.add(new MenuItem("Exit"));
mb.add(m1);
  Menu m2 = new Menu("Editar"); // Novo menu 2
mb.add(m2);
  Menu m3 = new Menu("?");      // Novo menu 3
mb.setHelpMenu(m3);
(...)
```

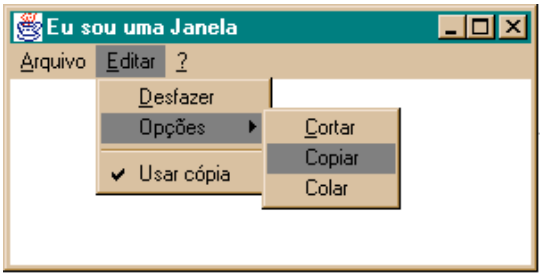


MenuItem

Esta classe define um item de menu com um determinado rótulo textual. O método de instância `add()` da classe `Menu` é utilizado para adicioná-lo a um menu. Como `Menu` também é um `MenuItem`, é possível incluí-lo dentro de outros

menus. `CheckBoxMenuItem` é uma especialização de `MenuItem`, que cria um item que pode ser ligado ou desligado. O exemplo a seguir ilustra essas opções:

```
(...)
  Menu m2 = new Menu("&Editar"); // Novo menu m2
    m2.add(new MenuItem("Desfazer"));
    Menu m4 = new Menu("Opções"); // Novo menu m4
      m4.add(new MenuItem("Cortar"));
      m4.add(new MenuItem("Copiar"));
      m4.add(new MenuItem("Colar"));
    m2.add(m4);     // m4 é sub-menu (item) de m2
    m2.addSeparator();
    m2.add(new CheckboxMenuItem("Usar cópia"));
mb.add(m2);
(...)
```



Para capturar e lidar com um evento produzido pela seleção de uma opção de menu deve-se usar a estrutura para capturar `ActionEvents`.

Menus PopUp

Em Java 1.0, menus só podiam ser adicionados em Frames. No JDK 1.1 se permite que menus pulem de qualquer parte de um componente. São os "pop-up menus". Para implementá-los basta instanciar a classe `PopupMenu`, acrescentar itens `MenuItem` ou submenus com a classe Menu e adicioná-los ao componente usando o seu método `add()`.

```
PopupMenu pop = new PopupMenu();     // cria menu
pop.add(new MenuItem("Recortar");          // adiciona itens
pop.add(new MenuItem("Colar");
pop.add(new MenuItem("Copiar");
pop.addSeparator();
pop.add(new Menu("Mover para"));
add(pop);  // adiciona ao componente
```

Para fazer o menu aparecer, é preciso capturar o evento que dispara o menu popup ("pop-up trigger") E, nele, chamar o método `show()` de `PopupMenu` nas coordenadas onde o mouse foi acionado:


```
public void processMouseEvent(MouseEvent e) {
if (e.isPopupTrigger()) {
    pop.show(this, e.getX(), e.getY());
 super.processMouseEvent(e);
}
```

A forma de implementação do `PopupTrigger` varia entre plataformas (no Windows, ele é disparado ao se clicar o botão direito do mouse).

Exercícios

2. Incremente a aplicação que você fez no primeiro exercício com menus e pop-up menus.

1.4. Layouts

As classes descritas a seguir implementam a interface `LayoutManager` ou `LayoutManager2`. Estas interfaces definem os métodos necessários para que uma classe possa arrumar objetos do tipo `Component` dentro de um objeto do tipo `Container`.

Layouts em Java são uma forma independente de plataforma de posicionar objetos em uma GUI (você pode usá-los se quiser, ou desligá-los, mas fazendo isto, você terá que posicionar os componentes definindo os pixels que compõem suas coordenadas).

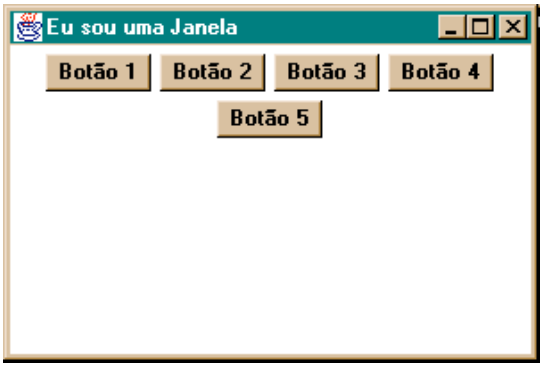
FlowLayout

Organiza os objetos da esquerda para a direita em linhas. Quando acaba uma linha, segue para a próxima. É usado por *default* na classe `Panel` (e pela sua subclasse `Applet`). Na classe `Frame` pode ser definido usando `setLayout(new FlowLayout())`. Exemplo:

```
public class Quadro extends Frame{

    public Quadro() {
        super("Eu sou uma Janela");
        setSize(300,200);

        setLayout(new FlowLayout());
        add(new Button("Botão 1"));
        add(new Button("Botão 2"));
        add(new Button("Botão 3"));
        add(new Button("Botão 4"));
```



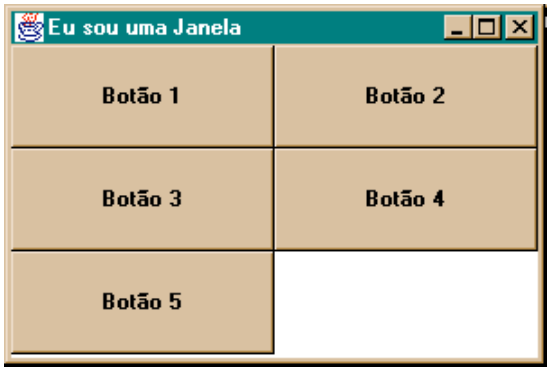
```
        add(new Button("Botão 5"));

        setVisible(true);
    }

    public static void main(String a[]) {
        new Quadro();
    }
}
```



GridLayout

Divide o recipiente em um número especificado de linhas e colunas e arruma os componentes neles da esquerda para a direita e de cima para baixo. Exemplo:

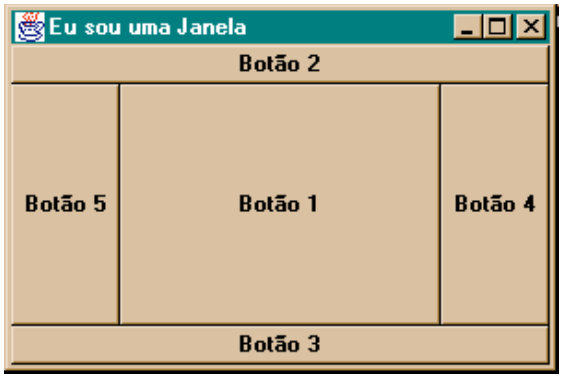
```
        setLayout(new GridLayout(3, 2));
        add(new Button("Botão 1"));
        add(new Button("Botão 2"));
        add(new Button("Botão 3"));
        add(new Button("Botão 4"));
        add(new Button("Botão 5"));

        setVisible(true);
```



BorderLayout

Arruma os componentes nas laterais do recipiente, usando os nomes `North`, `South`, `East`, `West` e `Center` ou as constantes `BorderLayout.NORTH`, `SOUTH`, `EAST`, `WEST` e `CENTER` no método `add()`, para informar a posição de cada objeto. Exemplo:

```
setLayout(new BorderLayout());
add("Center", new Button("Botão 1"));
add("North", new Button("Botão 2"));
add("South", new Button("Botão 3"));
add("East", new Button("Botão 4"));
add("West", new Button("Botão 5"));

setVisible(true);
```



CardLayout

Só é possível visualizar na tela um componente de cada vez, se eles forem organizados por `CardLayout`. O uso mais comum, é definir objetos `Panel` (possivelmente com *layouts* diferentes), cada qual com seus componentes e organizá-los com `CardLayout`. Pode-se, então, determinar botões específicos para mostrar cada um deles, um por vez. A applet a seguir mostra, dependendo do botão selecionado, um dos dois painéis mostrados na figura:

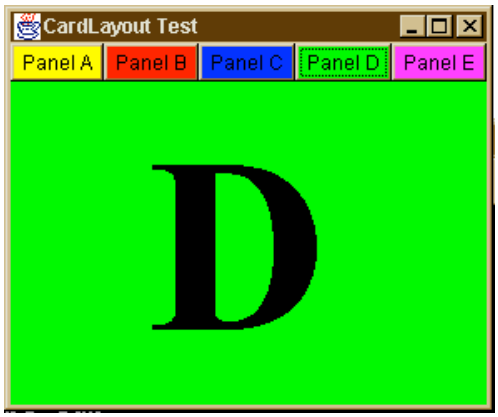
```
import java.awt.*;
import java.awt.event.*;

public class CardLayoutTest extends Frame {

   private Button b1, b2, b3, b4, b5;
   private CardLayout tabs;

   public CardLayoutTest() {

      super("CardLayout Test");
```



```
// Button's panel
Panel topPanel = new Panel(new FlowLayout(0,0,FlowLayout.LEFT));

b1 = new Button("Panel A");
b1.setBackground(Color.yellow);
b2 = new Button("Panel B");
b2.setBackground(Color.red);
b3 = new Button("Panel C");
b3.setBackground(Color.blue);
b4 = new Button("Panel D");
b4.setBackground(Color.green);
b5 = new Button("Panel E");
b5.setBackground(Color.magenta);

topPanel.add(b1);
topPanel.add(b2);
topPanel.add(b3);
topPanel.add(b4);
topPanel.add(b5);

add(BorderLayout.NORTH, topPanel);

// Card Panel
tabs = new CardLayout();
final Panel lowerPanel = new Panel(tabs);
lowerPanel.setFont(new Font("Helvetica", Font.BOLD, 150));

Panel p1 = new Panel();
p1.setBackground(Color.yellow);
p1.add(new Label("A"));

Panel p2 = new Panel();
p2.setBackground(Color.red);
p2.add(new Label("B"));

Panel p3 = new Panel();
p3.setBackground(Color.blue);
p3.add(new Label("C"));

Panel p4 = new Panel();
p4.setBackground(Color.green);
p4.add(new Label("D"));

Panel p5 = new Panel();
p5.setBackground(Color.magenta);
p5.add(new Label("E"));

lowerPanel.add("pan1", p1);
lowerPanel.add("pan2", p2);
lowerPanel.add("pan3", p3);
lowerPanel.add("pan4", p4);
lowerPanel.add("pan5", p5);
```

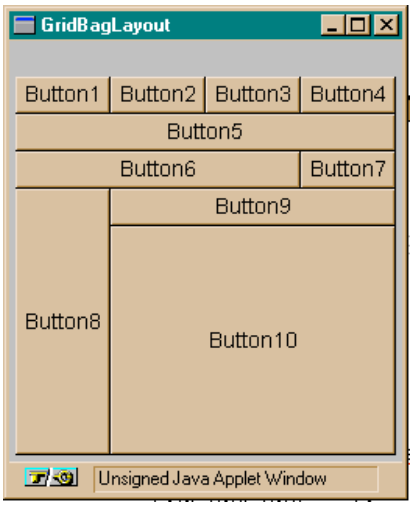


```
        add(BorderLayout.CENTER, lowerPanel);
        tabs.show(lowerPanel,"pan1");

        // eventos
        b1.addActionListener (new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              tabs.show(lowerPanel,"pan1");
           }
        });

        b2.addActionListener (new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              tabs.show(lowerPanel,"pan2");
           }
        });

        b3.addActionListener (new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              tabs.show(lowerPanel,"pan3");
           }
        });

        b4.addActionListener (new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              tabs.show(lowerPanel,"pan4");
           }
        });

        b5.addActionListener (new ActionListener() {
           public void actionPerformed(ActionEvent e) {
              tabs.show(lowerPanel,"pan5");
           }
        });

        // Frame
        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        new CardLayoutTest();
    }
}
```

GridBagLayout

É o mais complicado dos gerentes de layout. Divide o recipiente em uma grade de linhas e colunas, com dimensões variáveis definidas por um objeto `GridBagConstraints`. Com este gerente de layout, é possível organizar os componentes de uma maneira elegante e garantir que eles não se sobreponham. O `GridBagLayout` não será abordado neste curso.



Outros layouts

Usando os layouts disponíveis para organizar componentes em `Panels` e depois utilizar esses mesmos `Panels` em outros `Panels`, com outros layouts, é possível colocar componentes AWT onde você quiser. Você também tem a opção de não utilizar layout algum. Para isto é somente definir o layout como nulo:

```
setLayout(null);
```


1.5. Você terá então que posicionar cada objeto no recipiente, de acordo com as suas coordenadas em pixels. Terá também que definir os tamanhos e espaços entre os componentes. Quando se usa um layout, não existe liberdade. Eles são tirânicos. Não adianta usar `setSize()` em um `BorderLayout`, assim como não adianta usar um `setLocation()` em `FlowLayout`. Simplesmente nada acontece, pois as regras do algoritmo de posicionamento e redimensionamento do layout sempre têm prioridade. A opção de desligar os layouts, embora ofereça liberdade absoluta, diminui a portabilidade do seu programa, pois não é possível prever como a janela irá se apresentar quando exibida em outro sistema operacional.

