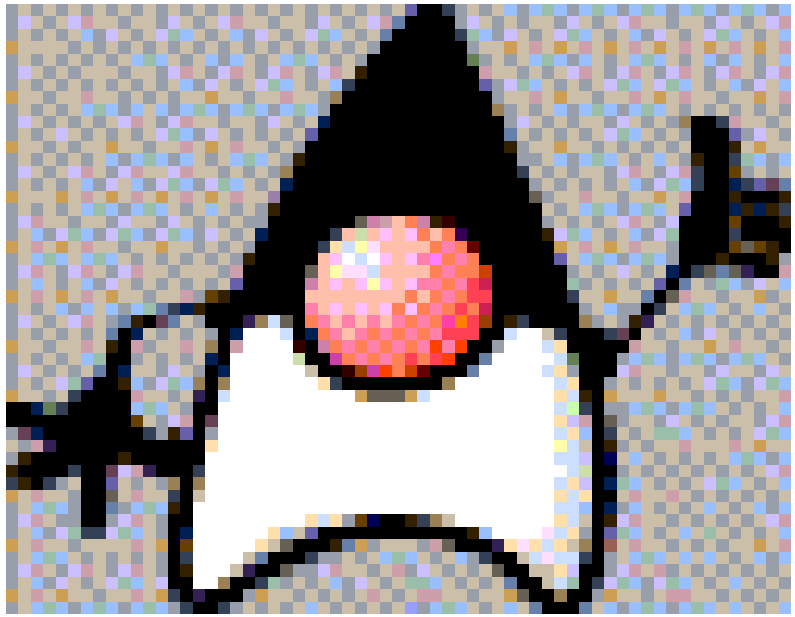
Interfaces Gráficas em Java



Introdução

- Java fornece um forte suporte para a construção de interfaces gráficas através do pacote java.awt
- GUI inicial do Java:
  - ⇨ incompleta (falta grid, por exemplo)
  - ⇨ com problemas de portabilidade (peers)
- Substituída pela JFC (swing), mas compartilham partes fundamentais (como eventos)

Elementos Gráficos

- Os elementos chaves de uma interface gráfica em Java são:
  - ⇨Componentes gráficos (botões etc.)
  - ⇨Gerenciadores de layoutes
  - ⇨Processamento de eventos
- Componentes gráficos, tais como campo texto e botões, são elementos que o usuário manipula com o mouse ou com o teclado
- Gerenciadores de layout governam a maneira pela qual os componentes aparecem na tela
- Eventos assinalam ações do usuário consideradas importantes, como o click de um mouse em cima de um botão

Programação Dirigida por Eventos

- Programas Java devem responder a eventos gerados por componentes gráficos indicando que ações específicas ocorreram sobre tais componentes
- Uma categoria especial de classes, chamadas *listeners*, que ficam a espera de eventos
- Portanto, uma programa cliente Java é, geralmene, composto por:
    - o código representando a interface gráfica com o usuário
    - *listeners* escutando eventos
    - código que respondem a eventos

Componentes Gráficos

- Awt contém vários tipos de componentes gráficos permitindo diferentes maneiras de interagir com o usuário:
  - ⇨labels
  - ⇨text fields
  - ⇨text areas
  - ⇨lists
  - ⇨buttons
  - ⇨scrollbars

Classes de suporte

Além das classes vistas no slide anterior, a interface gráfica necessita utilizar várias classes de auxílio à montagem de uma interface gráfica. São elas:

- **Point**
- **Dimension**
- **Rectangle**
- **Color**
- **Font**

Point

Propósito: representar um ponto num sistema de coordenadas (linhas e colunas) visando indicar a localização de um componente visual AWT no container onde o mesmo está inserido.

Construtores:

Point ( ) Cria uma ponto nas coordenadas (0, 0).

Point (int x, int y) Cria uma ponto através das coordenadas x e y especificadas.

Variáveis:

x- representa a coluna.

y- representa a linha

Point

Método	***Uso***
move(int x,int y)	movimenta o ponto para uma nova localização.
translate(int dx,int dy)	movimenta o ponto, incrementando suas coordenadas x e y com os valores de dx e dy respectivamente.

Dimension

Propósito: representar a dimensão (em largura e altura) de um componente visual AWT.

Construtores:

Dimension ( ) Cria uma dimensão com 0 para largura e altura.

Dimension (int largura, int altura)
Cria uma dimensão com os valores definidos nos parâmetros para largura e altura.

Variáveis:

- height - a altura da dimensão.
- width - a largura da dimensão.

Dimension

Método	***Uso***
setSize(int l,int a)	redefine valores para a largura e altura de uma dimensão.
equals(Object d)	verifica se dois objetos da classe Dimension têm os mesmos valores para altura e largura.

Rectangle

Propósito: representar um retângulo que especifica o formato (em largura e altura) e a localização (eixos x e y) de um componente visual AWT.

Construtores:

Rectangle (int x, int y, int largura, int altura)

Cria um retângulo com localização e formato pré-definidos.

Rectangle (Point p, Dimension d)

Cria um retângulo com localização representada pelo objeto *p* e formato representado pelo objeto *d*.

Variáveis:

- x - representa a coluna.
- y - representa a linha.
- height - a altura do retângulo.
- width - a largura do retângulo.

Rectangle

Método	***Uso***
getLocation( )	retorna um objeto Point representando a localização do retângulo.
setLocation(Point p)	redefine a localização do retângulo.
contains(Point p)	verifica se o ponto p está contido na área do retângulo.
grow(int l, int a)	aumenta o retângulo com base nos valores de incremento l e ª
intersection(Rectangle r)	calcula a interseção entre dois retângulos devolvendo um novo.

Color

Propósito: representar uma cor através da composição das cores primárias vermelho, verde e azul. A cor gerada poderá ser então atribuída a um componente visual, para definir suas propriedades para cor de frente e fundo.

Construtor: Color(int red, int green, int blue)

⇨ Cria uma cor com a composição dos componentes vermelho , verde e azul.

Variáveis:

black	blue	cyan	darkGray
gray	green	lightGray	magenta
orange	pink	red	white
yellow			

Color

Método	*Uso*
brighter( )	Cria uma versão mais clara da cor.
darker( )	Cria uma versão mais escura da cor.
getBlue( )	Retorna um inteiro correspondente ao componente azul.
getGreen( )	Retorna um inteiro correspondente ao componente verde.
getRed( )	Retorna um inteiro correspondente ao componente verde.

Font

Construtor

```
Font(String name, int style, int size)
```

Atributos

```
PLAIN, BOLD, ITALIC
```

Exemplo

```
Font f = new Font("arial", Font.PLAIN,14);
```

Criando aplicações gráficas AWT e JFC

- *A criação de interfaces gráficas AWT consiste basicamente na criação (instância) de objetos do tipo Conponent (botões, textos etc.), na criação de recipientes ou objetos da classe Container (janelas, paineis etc.) para receber os componentes criados e na adição dos componentes aos recipientes, com base num sistema de coordenadas (especificando a localização da inserção) ou via utilização de gerentes de layouts que se encarregam de definir a localização e aspecto visual dos componentes inseridos nos recipientes.*

Component

- *A classe Component representa um objeto que tenha uma representação gráfica, como botões, campos de textos, choices etc. Esta classe define o comportamento básico para a maioria dos componentes visuais do pacote AWT (todos os métodos definidos nesta classe estarão disponíveis para todos os componentes visuais do AWT, bem como todos os recipientes).*

- *Ex.: método setVisible(boolean visibilidade)*
  - *botao.setVisible(true); // faz um Button aparecer no Container*
  - *janela.setVisible(false); // faz um Frame desaparecer*

Component

Método	***Uso***
getName( )	Retorna o nome do componente
setName(String n)	Atribui um nome ao componente
getParent( )	Retorna o Container onde o componente foi adicionado
isVisible( )	Determina se o componente é visível. Componentes são inicialmente visíveis, exceto Frame, Window e Dialog.
setVisible(boolean b)	Torna o componente visível ou não, de acordo com o parâmetro

Component

Método	***Uso***
isShowing( )	Retorna true se o componente estiver dentro de um outro componente que está for visível e sendo mostrado.
isEnabled( )	Determina se um componente está habilitado para receber a entrada do usuário e gerar eventos. Componentes são inicialmente habilitados.
setEnabled(boolean b)	Habilita ou desabilita o componente.

Component

Outros métodos (aparência):

public Color getForeground( )
public void setForeground(Color c)
public Color getBackground( )
public void setBackground(Color c)
public Font getFont( )
public synchronized void setFont(Font f)

Component

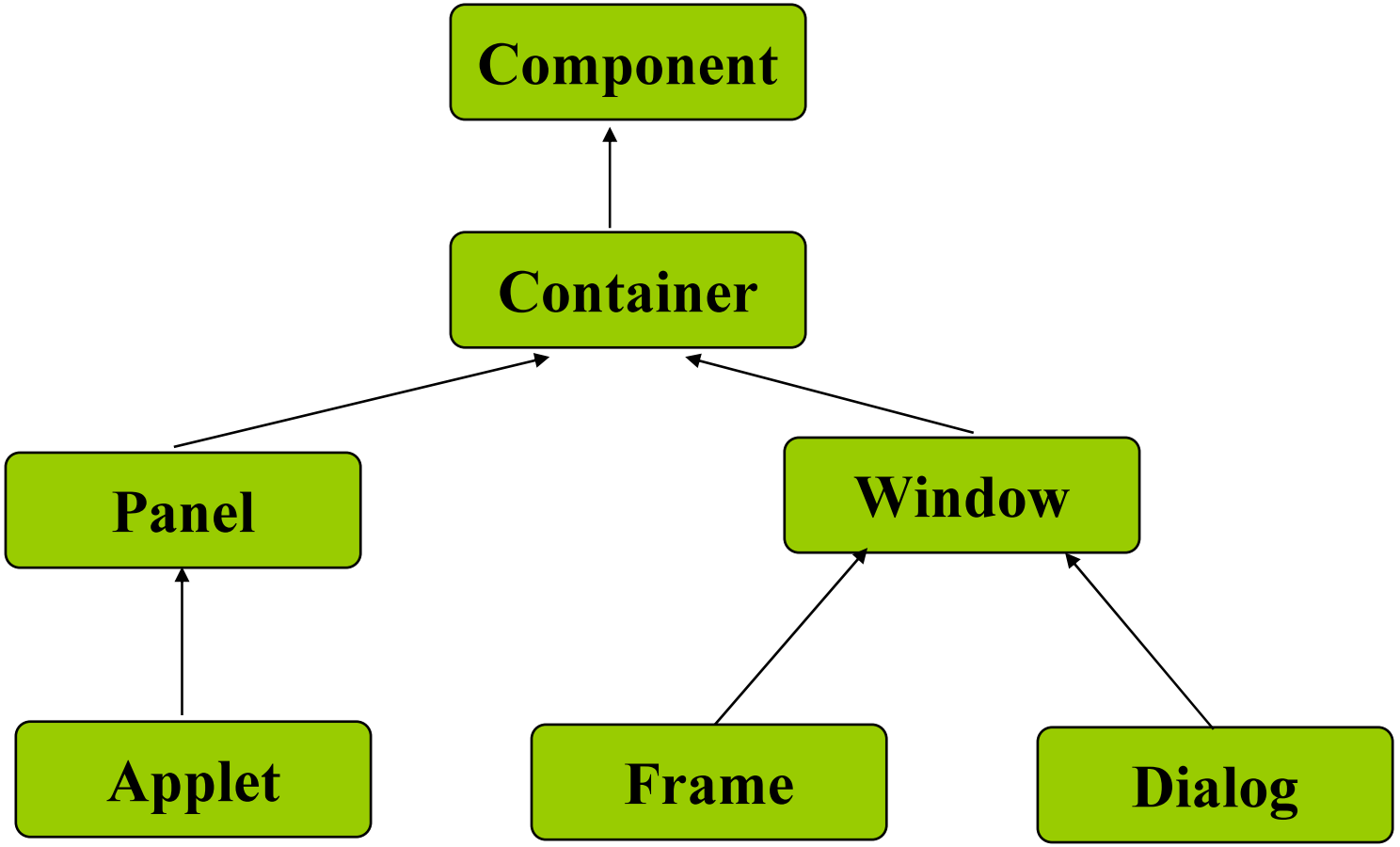
Outros métodos (localização):

public Point getLocation( )
public Point getLocationOnScreen( )
public void setLocation(int x, int y)
public void setLocation(Point p)
public void setBounds(int x,int y,int largura,int altura)
public void setBounds(Rectangle r)
public Dimension getSize( )
public void setSize(int width, int height)
public void setSize(Dimension d)

Containers

- Um *container* é uma categoria especial de componente gráfico que pode conter outros componentes ou mesmo outros containers.
- Todos os containers são componentes, mas nem todos os componentes são containers.
- Uma applet, uma janela e um painel são exemplos de container.
- Um container (painel) pode ser inserido dentro de outro.
- Cada container possui associado um gerenciador de layout para controlar a maneira pela qual seus componentes serão mostrados (tamanho e posição).

Containers do pacote AWT



Containers

- Alguns containers devem ser associados a uma outra superfície gráfica:
  - ⇨panel
  - ⇨applet
- Uma applet é associada a um browser ou a uma janela do appletviewer e um panel é associado a qualquer outro container.
- Outros containers podem ser movimentados de maneira independente:
  - ⇨window
  - ⇨frame
  - ⇨dialog

Container - métodos

- add(Component c)
  - Adiciona um componente ao container.
- add(Component c, int i)
  - Adiciona um componente ao container na posição indicada pelo segundo parâmetro.
- addContainerListener(ContainerListener c)
  - Associa um “ouvinte” de eventos para o container.
- doLayout( )
  - Reorganiza os componentes do container.

Container - métodos

- **getAlignmentX( )**
  - Retorna um inteiro correspondente ao alinhamento no eixo x.
- **getAlignmentY( )**
  - Retorna um inteiro correspondente ao alinhamento no eixo y.
- **getComponent(int i)**
  - Retorna o componente localizado na posição indicada pelo parâmetro fornecido.
- **getComponentAt(int x, int y)**
  - Localiza o componente que contêm a posição x,y.

Container - métodos

- getComponentCount()
  - Retorna o número de componentes inseridos.
- getComponents( )
  - Devolve um array contendo todos os componentes inseridos no container.
- getLayout( )
  - Devolve o gerente de layout utilizado pelo container.
- isAncestorOf(Component c)
  - Verifica se o componente está contido hierarquicamente no Container.
- remove(Component c)
  - Remove o componente especificado do container.

Container - métodos

- remove(int)
  - Remove o componente especificado pelo índice do container.
- removeAll( )
  - Remove todos os componentes inseridos.
- setCursor(Cursor c)
  - Redefine a imagem associada ao cursor dentro do container.
- setLayout(LayoutManager l)
  - Redefine o gerente de layout para o componente.
- validate( )
  - Reorganiza o container e todos os seus componentes.

Window

- *Uma Window é uma janela sem barra de título e borda borda e que necessariamente tem que estar associada com outro objeto da classe Frame (janela) para que possa existir.*
- *A classe Window especializa a classe container, logo poderá conter outros componentes internamente.*
- *A classe Window normalmente é utilizada para implementar janelas pop-up.*
- *Quando um objeto da classe Window é criado, automaticamente é associado a ele o gerente de layout BorderLayout.*

Window

Construtor

Window(Frame f)

Constrói um objeto (instância) da classe Window invisível que estará vinculado ao objeto Frame (janela) previamente instanciado.

Window

Método	***Uso***
show( )	Mostra a janela e traz ela para a frente das outras.
toBack( )	Move a janela para traz de outra janela.
toFront( )	Move a janela para frente de outra janela.
pack( )	Redimensiona a janela de tal forma que todos os seus componentes fiquem com seus tamanhos

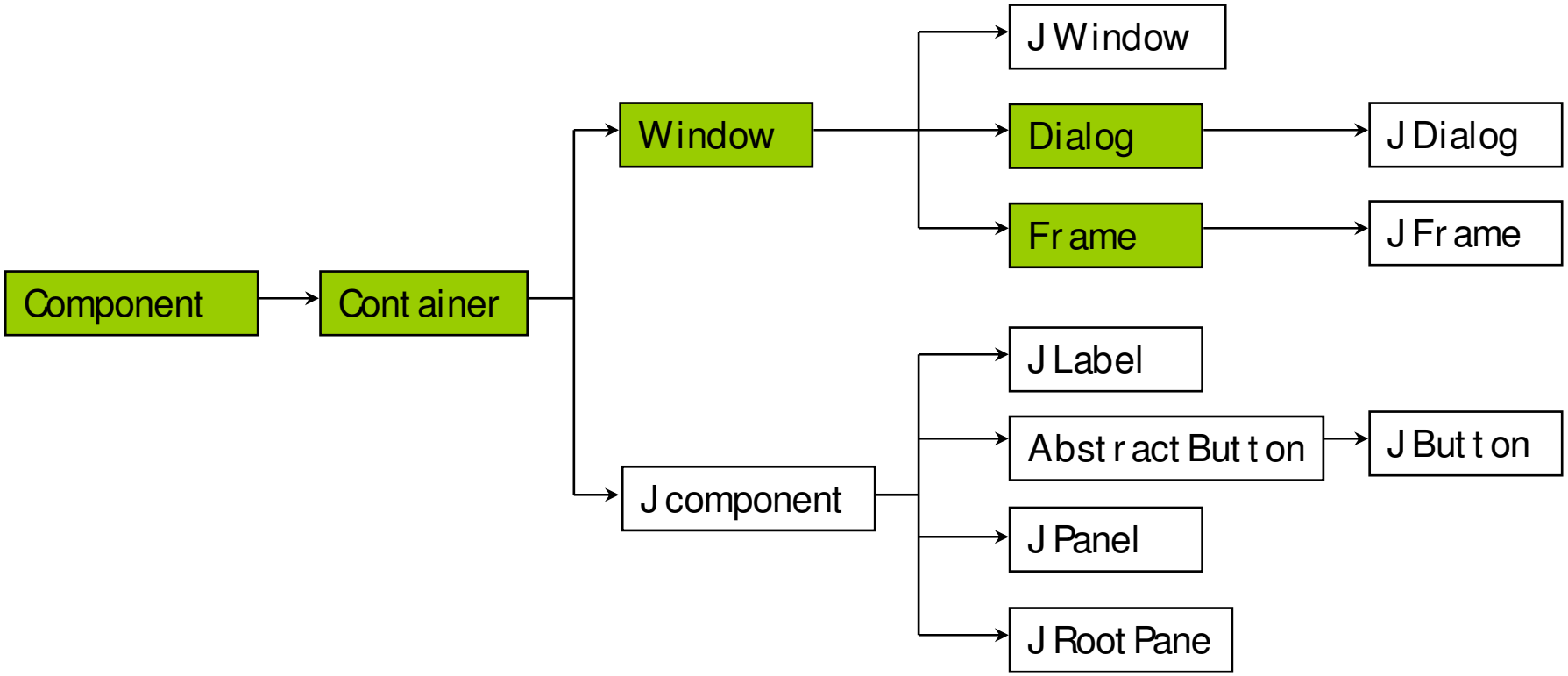
Migrando de AWT para Swing

- A AWT sempre esteve presente na linguagem Java. Os seus defeitos a muito tempo são bem conhecidos e os desenvolvedores tem gasto muito tempo e esforço na criação de componentes para prover alguma funcionalidade inexistente na AWT para construir interfaces que atentam às necessidades dos usuários.
- A meta do projeto Swing foi acrescentar novas funcionalidades, através de uma biblioteca de classes, a fim de suprir as demandas dos usuários.

O que é o JFC ?
(Java Foundation Class)

- É um conjunto de aspectos (componentes) que foram implementados e disponibilizados em uma biblioteca e serve para criar interfaces gráficas de usuário.
- Iniciou sua implementação no JDK1.1 e continua nas versões seguintes.

A estrutura hierárquica AWT e Swing



- *Os retângulos sombreados representam os controles da AWT.*

Qual a relação entre JFC e AWT ?

- Com a introdução de um novo conjunto de componentes para construção de interface que usa um modelo diferente daquele da AWT é natural perguntar-se o que fazer com a AWT.
- Para conviver Swing e AWT, a versão JDK1.1 foi alterada e introduzida uma nova facilidade denominada de “Lightweight components”- (componentes leves).
  - Significa: que os desenvolvedores possam diretamente estender uma classe **Component e Container** e utilizar as facilidades disponíveis nestas classes.

Qual a relação entre JFC e AWT ?

- Em JFC1.1 e JDK1.2, os componentes AWT e os componentes Swing são ambos suportados, embora os componentes AWT continuem usando o “peer model”. E não se sabe até quando esta situação manter-se-á.
- Alguns dos componentes Swing, atualmente, são derivados de componentes AWT.
  - P.ex: o componente JFrame, que fornece a janela principal da aplicação, é derivado da classe Frame da AWT.

Qual a relação entre JFC e AWT ?

- Todos os componentes AWT possuem um componente correspondente no Swing, logo é possível implementar aplicações que não dependam diretamente da AWT.

Nota: Existe uma tendência da Javasoft em não usar o "peer model". Logo, os desenvolvedores devem ficar atentos quanto a migrarem as aplicações existentes em AWT para o Swing.

O que necessito reaprender para usar os componentes Swing?

- É possível utilizar alguns componentes básicos do Swing diretamente.
- Com o Swing não há mudanças fundamentais no modo como as aplicações são construídas.
- Pode-se criar a janela principal da aplicação, usar *frames*, componentes e gerenciadores de layout e é possível estabelecer conexão quase que da mesma forma.

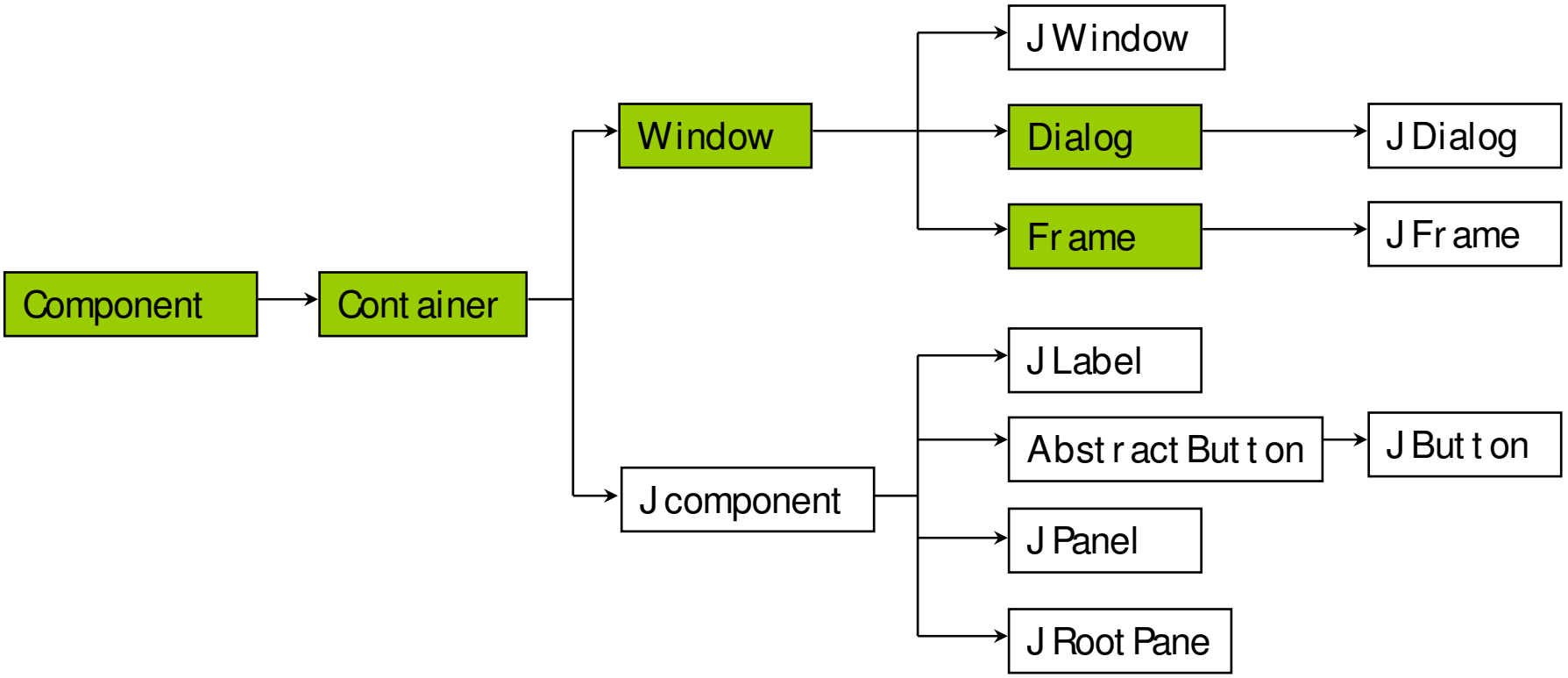
Os pacotes do Swing

- O produto JFC 1.1 é uma coleção de pacotes que contém os componentes do Swing, e que pode ser adicionado ao JDK 1.1.
- Para o JDK1.1, o nome dos pacotes iniciam com *java.awt.swing*.
- Em JDK1.2, os mesmos pacotes existem, porém o nome dos pacotes foi mudado para *javax.swing*.
- Segue a relação dos pacotes que são comuns a JFC1.1 e JDK1.2, agrupados por funcionalidade.

Os pacotes do Swing

javax.swing
javax.swing.border
javax.swing.event
javax.swing.plaf
javax.swing.plaf.basic
javax.swing.plaf.metal
javax.swing.plaf.motif
javax.swing.plaf.multi
javax.swing.plaf.windows
javax.swing.preview
javax.swing.table
javax.swing.text
javax.swing.text.html
javax.swing.text.rtf
javax.swing.tree
javax.swing.undo

A estrutura hierárquica AWT e Swing



- *Os retângulos sombreados representam os controles da AWT.*

A classe Jcomponent fornece:

- Uma aparência que (”a pluggable look and feel- l&f”) pode ser especificado pelo programador ou (opcionalmente) selecionada pelo usuário em tempo de execução.
- Criação de componentes personalizados.
- Uma borda que implicitamente define os limites do componente.
- A habilidade para definir o tamanho máximo, mínimo e preferido de um componente.
- ToolTips-- descrições curtas quando o cursor passa sobre um componente.
- Autoscrolling-- ocorre a rolagem automática em lista, tabelas ou árvore quando o mouse é arrastado sobre estes com o botão pressionado.

A classe Jcomponent

Método	Uso
`setToolTipText(String text)`	**registra o texto para ser mostrado em tooltip.**
`createToolTip()`	**retorna o JToolTip que deve ser usado para mostrar um texto tooltip**
`getBorder()`	**retorna a borda do componente ou nulo se nenhuma borda estiver definida.**
`setBorder()`	**atribui a borda do componente.**
`setPreferredSize(Dimension d)`	**atribui o tamanho preferido.**
`getPreferredSize()`	**retorna tamanho preferido.**

Definição de cores do componente

- Todos os componentes possuem dois modos de cores associados: a de fundo (background) e a de frente (foreground).
- A JFC introduz um conceito de um componente “lightweight”, que possui um fundo transparente que permite ver o que está atrás do componente. O componente transparente não faz uso de cores de fundo.
- A classe JComponent possue os métodos isOpaque, que retorna false se o componente é transparente, e setOpaque(boolean b), que permite alterar esta propriedade.

Frame

- Uma *Frame* é uma janela com uma *barra de título* e uma *borda*.
- A classe Frame especializa a classe Window, que por sua vez, especializa a classe container.
- A classe Frame implementa a interface MenuContainer, logo uma frame pode ter uma barra de menu associada a ela.
- Frames são geralmente usadas para construir aplicações, mas também podem ser usadas com applets.

Frame

- Se uma aplicação tem uma janela dependente de outra (que desaparece quando a outra é iconificada, por exemplo), então deve-se utilizar Dialog ou Window para esta janela.
- A classe Frame por padrão não fornece ao usuário mecanismos para ser fechada com o clique do mouse no botão para fechar a janela (**x**).

Frame

Construtor	***Operação***
Frame() Frame	Constrói uma nova instância de que é inicialmente invisível.
Frame(String t) Frame o	Constrói uma nova instância de que é inicialmente invisível, com título passado como parâmetro.

Frame

Método	***Uso***
getTitle()	Retorna um objeto String contendo o título.
setTitle(String t)	Atribui um título, passado como parâmetro, ao objeto Frame.
getIconImage() Image mostrada janela é iconificada.	Retorna um objeto do tipo contendo a imagem quando a
setIconImage(Image I)	Atruibui uma imagem a ser mostrada quando a janela é iconificada.

Frame

Método	***Uso***
getMenuBar()	Retorna a barra de menu do Frame
setMenuBar(MenuBar m)	Atribui uma barra de menu ao Frame.
isResizable()	Retorna `true` se o usuário pode modificar o tamanho do Frame e `false`, caso contrário

Frame

Método ***Uso***

setResizable(boolean b) Permite, ou não, que o usuário modifique o tamanho do Frame.

setLayout (LayoutManager l) especifica um novo gerente de layout para a janela (ou sem layout se for passado o valor null.

Ex.: objJanela.setLayout (null).

Frame

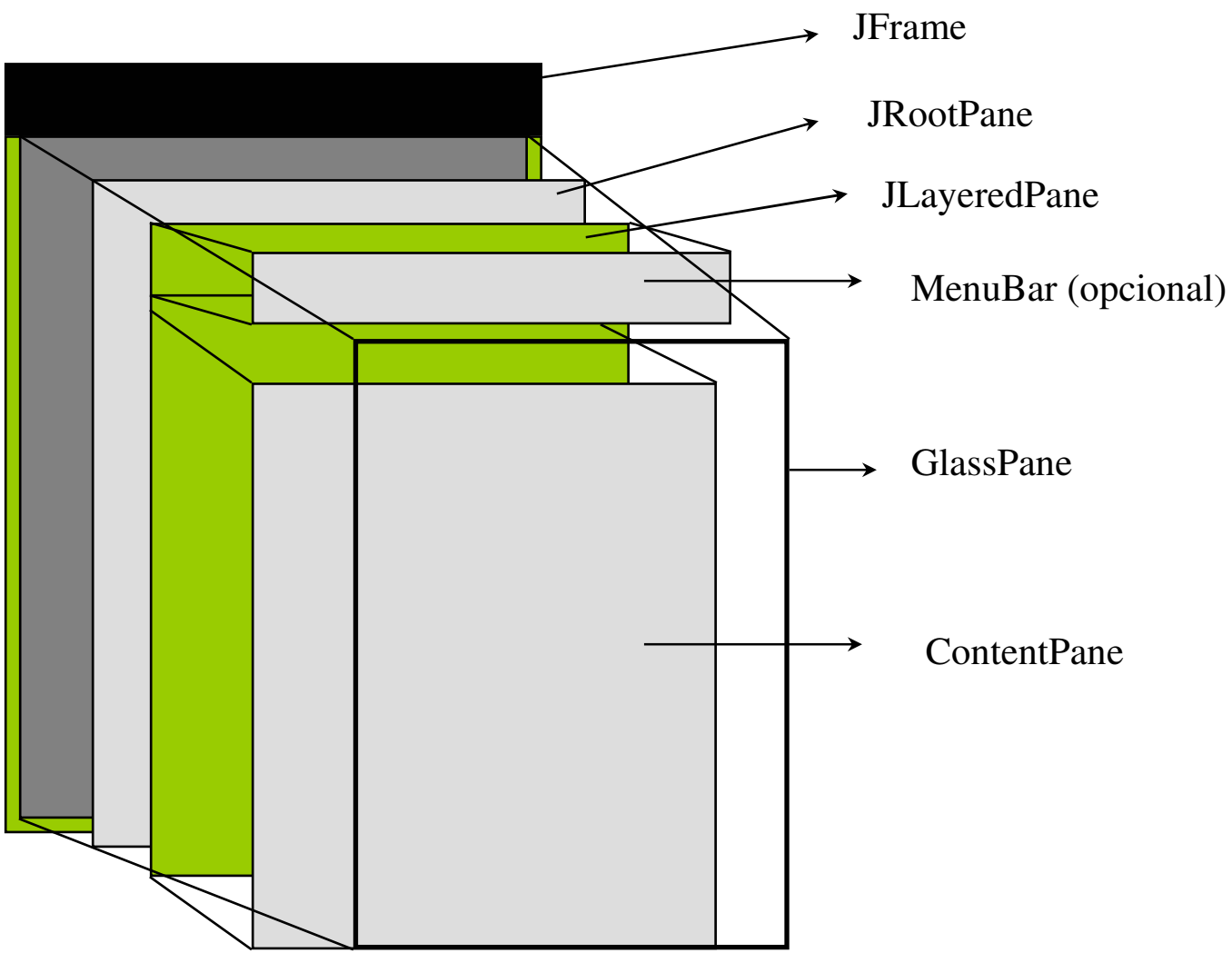
- Deve-se usar pack( ) ou setSize(int,int) em um Frame antes que ele seja mostrado pela primeira vez. Caso contrário, somente a barra de título da janela aparecerá na tela.
- Em geral, preferível utilizar o pack do que o setSize:
  - ⇨ pack delega para o gerenciador de layout a mudança no tamanho da janela.
  - ⇨ com o gerenciador de layout adquire-se melhor independência de plataforma.

A classe JFrame

o esqueleto de uma aplicação

- Quando um programa java inicia com uma GUI, vê-se a janela onde a aplicação irá fazer as interações com o usuário através de teclado e mouse. No Swing, a interface principal da aplicação é implementada pelo componente javax.swing.JFrame.
- Assim como outros componentes, o JFrame é derivado do controle da AWT. Segue a estrutura hierárquica dos componentes AWT e Swing.

A estrutura de JFrame



A estrutura de JFrame

- Quando criado, o JFrame possui um único filho o container que é uma instância da classe JRootPane.
- A classe JRootPane pode gerenciar outros 4 componentes: um JLayeredPane, um ContentPane, um MenuBar e um GlassPane.
- JRootPane é gerenciado de forma a cobrir a superfície inteira da JFrame, exceto a barra de título e o tamanho da barra.

A estrutura de JFrame

- JLayeredPane : adiciona profundidade a um container, permitindo sobreposição de componentes quando necessário.
- MenuBar: a classe MenuBar class encapsula os conceitos da plataforma para os limites da barra de menu para uma janela. O objetivo é associar a barra de menu com um objeto Frame, chamando o método setMenuBar.

JFrame

- A classe JFrame é ligeiramente incompatível com java.awt.Frame. JFrame contém a classe JRootPane como filho único.
- A **contentPane** deve ser pai de qualquer filho da classe JFrame. Essa é diferença da java.awt.Frame. P.ex. para adicionar um filho na Frame da AWT deve-se escrever:

  frame.add(child);
- Contudo, usando a JFrame necessita-se adicionar o filho ao contentPane da JFrame:

  frame.getContentPane().add(child);

JFrame

Método	***Uso***
getContentPane( )	retorna o objeto contentPane para este Frame.
getGlassPane( )	retorna o objeto glassPane para a janela.
getJMenuBar( )	retorna a barra de menu da janela.
getJLayredPane( )	retorna o objeto layeredPane para a janela.

JFrame

Método ***Uso***

void **setDefaultCloseOperation**(int operação) : determina a operação default para quando o usuáro tentar fechar a janela.

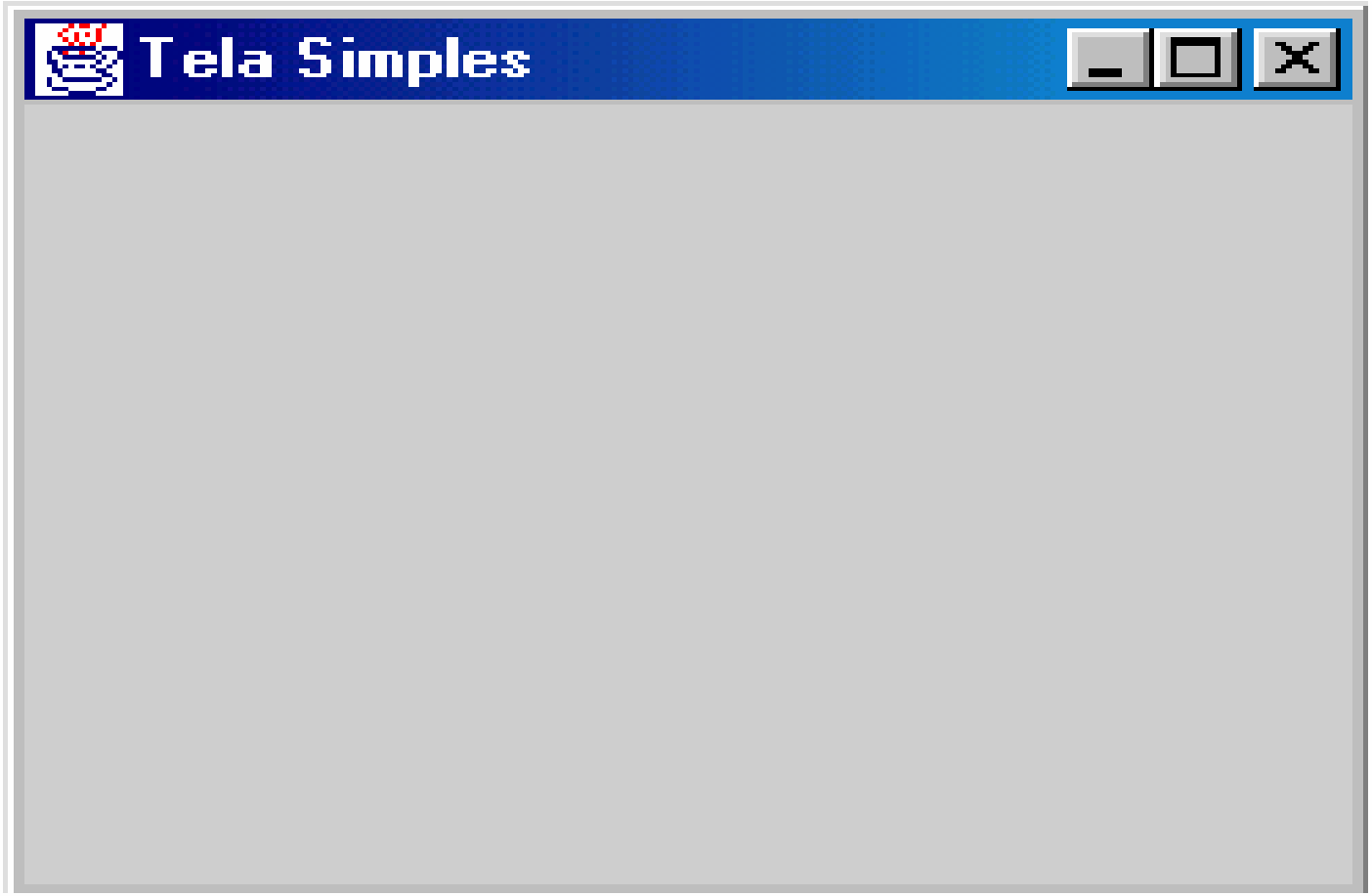
Tipos de operação:

- JFrame.DO_NOTHING_ON_CLOSE (definido em WindowConstants): nada acontece, pois requer que o programa manipule a operação no método windowClosing do evento WindowListener registrado pelo objeto que gerou a ação.
- JFrame. HIDE_ON_CLOSE (definido em WindowConstants): automaticamente oculta a janela após ser invocado qualquer objeto WindowListener. Este é o valor default de JFrame.
- JFrame. DISPOSE_ON_CLOSE (definido em WindowConstants): automaticamente oculta e descarta a janela após ser invocado qualquer objeto WindowListener.
- JFrame. EXIT_ON_CLOSE (definido em JFrame): automaticamente sai da aplicação utilizando o método exit( ) da classe System. Aconselha-se que esta operação seja usada apenas em aplicações.

JFrame - exemplo

```
import javax.swing.*;
public class BasicFrame extends JFrame {
   public static void main(String[] args) {
        BasicFrame bf = new BasicFrame ("Tela Simples");
     }

   BasicFrame (String title) {
       super(title);
       setSize(250, 200);
       setVisible(true);
   }
}
```

JFrame - resultado



JFrame – exemplo 1

```
import javax.swing.*;
Import java.awt.*;
public class MyJFrame {

   public static void main (String args[]) {
      JFrame tela = new JFrame("Tela Exemplo 1");

      tela.setSize(200,200);
      tela.setBackground(Color.blue);
      tela.setVisible(true);
   }
}
```

JFrame – exemplo 2

```
import java.awt.*;
import javax.swing.*;
public class MyJFrame extends JFrame
{
   public static void main (String args[])
   {
      MyJFrame tela = new MyJFrame();
      tela.setTitle ("Tela Exemplo2");
      tela.setSize(500,500);
      tela.setBackground(Color.green);
      tela.setVisible(true);
   }
}
```

JFrame – exemplo 3

```
import java.awt.*;
import javax.swing.*;
public class MyJFrame extends JFrame
{
   public static void main (String args[])
   {
      MyJFrame tela = new MyJFrame("Tela Exemplo 3");
      tela.setSize(500,500);
      tela.setBackground(Color.lightGray);
      tela.setVisible(true);
   }
   MyJFrame(String titulo)
   {
      super(titulo);
   }
}
```

JFrame – exemplo 4

```
// Arquivo MyJFrame.java
import java.awt.*;
import javax.swing.*;
public class MyJFrame extends JFrame
{
    MyJFrame(String titulo)
    {
       super(titulo);
       setSize(200,200);
       setBackground(Color.blue);
       setVisible(true);
     }
}

// Arquivo ChamaMyJFrame.java
public class ChamaMyJFrame
{
   public static void main(String args[])
   {
      MyJFrame tela = new MyJFrame("Minha Tela");
   }
}
```

JFrame – exemplo 5

```
import java.awt.*;
import javax.swing.*;
public class MyJFrame extends JFrame
{
   public static void main (String args[])
   {
      MyJFrame tela = new MyJFrame();
      tela.setTitle ("Tela1 Exemplo 5");
      tela.setSize(500,500);
      tela.setBackground(Color.green);
      tela.setVisible(true);

      MyJFrame tela2 = new MyJFrame();
      tela2.setTitle ("Tela2 Exemplo 5");
      tela2.setSize(300,300);
      tela2.setBackground(Color.white);
      tela2.setVisible(true);
   }
}
```

JFrame – exemplo 6

```
import java.awt.*;
import javax.swing.*;
public class MyJFrame extends JFrame
{
   public static void main (String args[])
   {
      MyJFrame tela = new MyJFrame();
      tela.setTitle ("Tela1 Exemplo 6");
      tela.setSize(300,300);
      tela.setBackground(Color.green);
      tela.setVisible(true);

      MyJFrame tela2 = new MyJFrame();
      tela2.setTitle ("Tela2 Exemplo 6");
      tela2.setSize(500,500);
      tela2.setBackground(Color.white);
      tela2.setVisible(true);
      tela2.toBack();
   }
}
```

JFrame – exemplo 7

```
import java.awt.*;
import javax.swing.*;
public class MyJFrame extends JFrame {
   public static void main (String args[]) {
      MyJFrame tela2 = new MyJFrame();
      tela2.setTitle ("Tela Exemplo 7");
      tela2.setSize(500,500);
      tela2.setBackground(Color.white);
      tela2.setVisible(true);

      JWindow tela = new JWindow(tela2);
      tela.setSize(300,300);
      tela.setBackground(Color.green);
      tela.setVisible(true);
   }
}
```

JFrame – exercício

Durante a abordagem dos componentes e recipientes do pacote swing será utilizado um exercício único que, na conclusão do mesmo, deverá ter o aspecto mostrado no slide a seguir. A cada nova etapa (a cada novo componente aprendido) será utilizado o exercício anterior para inserção do componente adicional. Neste ponto, apenas criaremos a janela (recipiente) que conterá os demais componentes.

JFrame – exercício

Crie uma classe em java (CandidatoEmprego.java) que contenha um atributo privado de nome **janela** (um objeto da classe JFrame). Esse atributo deverá ser instanciado no construtor da classe atual:

Título:	Candidato a Emprego
Cor de Fundo:	(202,202,202)
Dimensão :	largura = 510 e altura = 490
Layout:	nulo
Resizable:	false

A classe CandidatoEmprego deverá conter um método público com a seguinte assinatura: ***public void mostraJanela().*** Este método se encarregará de tornar visível o objeto **janela** para o usuário.

*Incluir as instruções **import java.awt.*** .
import javax.swing.* .

JFrame – resultado final

JPanel

- A classe JPanel (painel) é uma sub-classe de JComponent.
- Tem o propósito de agrupar componentes para serem posteriormente inseridos em outro container.
- Permitem a criação de layouts sofisticados.
- podem conter componentes incluindo outros painéis.
- Utiliza o gerente de layout FlowLayout como padrão.

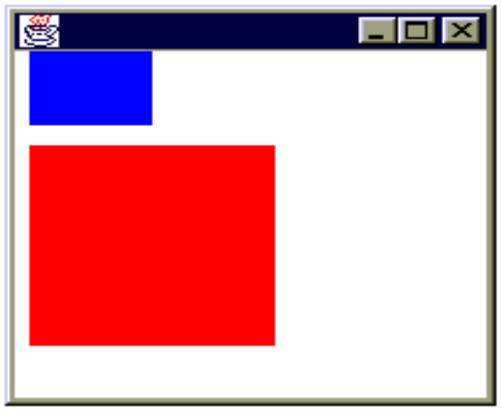
JPanel

Construtor	*Operação*
`JPanel()`	Constrói uma nova instância de um painel que é inicialmente invisível.
`JPanel(LayoutManager l)`	Constrói uma nova instância de um painel contendo um gerente de layout específico.

JPanel

- O exemplo a seguir mostra a criação de um panel que pode ser usado para inserir novos componentes.
- Obs.: lembre-se que um panel é uma subclasse da classe JComponent, logo poderá utilizar o método add(Component c) para inserir algum componente internamente (o mesmo vale para objetos da classe JFrame !).

JPanel - exemplo



```
import java.awt.*;
import javax.swing.*;
public class PanelDemo {
   private JPanel painelAzul;
   private JPanel painelVermelho;
   private JFrame janelaDemo;
   public PanelDemo()   {
      painelAzul = new JPanel();
      painelVermelho = new JPanel();
      janelaDemo = new JFrame();
      painelAzul.setBackground(Color.blue);
      painelVermelho.setBackground(Color.red);
      painelAzul.setSize(50,50);
      painelVermelho.setSize(100,100);
      janelaDemo.getContenPane().setLayout(null);
      janelaDemo.setSize(200,200);
      janelaDemo.getContentPane().add(painelAzul);
      janelaDemo.getContentPane().add(painelVermelho);
      painelAzul.setLocation(10,10);
      painelVermelho.setLocation(10,70);
      janelaDemo.setVisible(true);
   }
   public static void main(String[] args)   {
      PanelDemo p = new PanelDemo();
   }
}
```

JPanel – exercício (1)

Utilize a classe CandidatoEmprego.java e lhe acrescente 4 atributos privados da classe JPanel. Esses atributos deverão ser instanciados no construtor da classe atual:

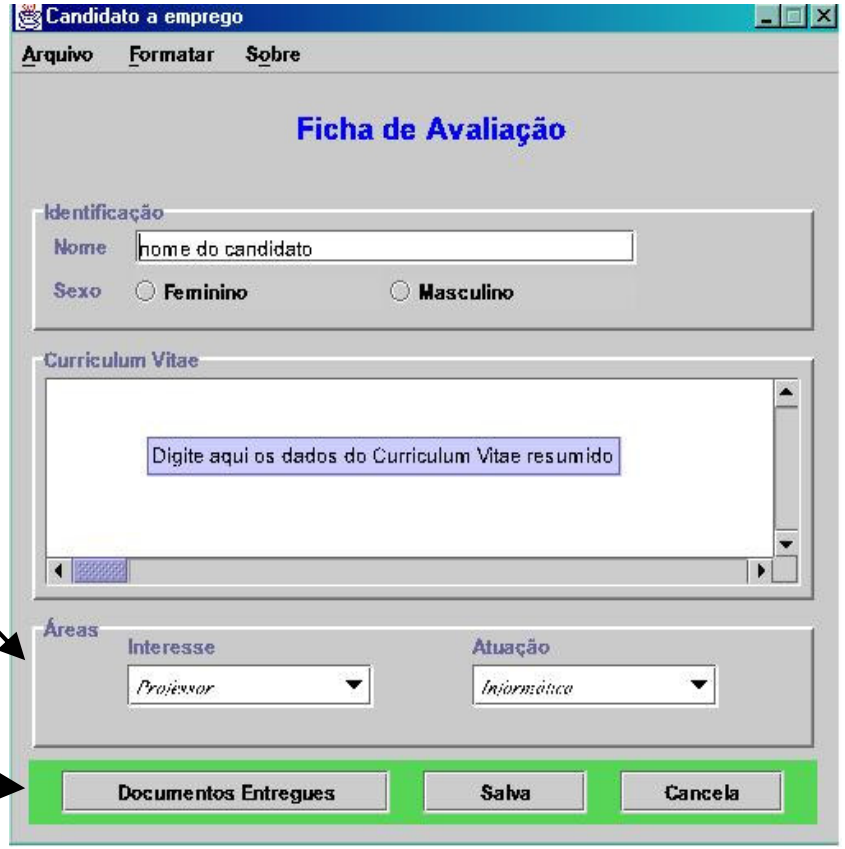
painelIdentificacao:
Cor de Fundo: (202,202,202)
Retângulo:(10,70,480,80)
Layout: nulo

painelCurriculum:
Cor de Fundo: (202,202,202)
Retângulo:(10,155,480,150)
Layout: nulo

J Panel – exercício (*2)

painelAreas:
Cor de Fundo: (202,202,202)
Retângulo:(10,310,480,80)
Layout: nulo

painelBotoes:
Cor de Fundo: (87,213,87)
Retângulo:(10, 395, 480,37)
Layout: nulo



*Adicioná-los ao ContentPane.

JDialog

- Um `JDialog` é uma janela com uma barra de título.
- Caixas de diálogos são em geral usadas para obter/mostrar informações do/para o usuário.
- O swing fornece um rico conjunto de diálogos que permite interações básicas com o usuário sem a necessidade de escrever muito código.

JDialog

- Caixas de diálogos podem ser modais ou não-modais
  - Caixas modais não permitem que outras janelas sejam acessadas até que a caixa de dialogo seja fechada.
    - Ex.: Janela de confirmação de exclusão de arquivos no Windows Explorer do Windows 9x
  - Caixas não-modais permitem que outras janelas sejam manipuladas concomitantemente a janela de dialogo
- BorderLayout é o gerente de layout usado implicitamente por caixas de diálogos
- JDialog utiliza uma estrutura de camadas como a classe JFrame. Assim, ao adicionarmos um componente, devemos utilizar o ContentPane:

JDialog

Construtor	*Operação*
`JDialog(Dialog f)`	Constrói uma nova instância de JDialog inicialmente invisível e vinculada a janela f.
`JDialog(Dialog f, boolean m)`	Constrói uma nova instância de JDialog inicialmente invisível, definida como modal caso o parâmetro m seja verdadeiro.
`JDialog(Dialog f, String titulo, boolean m)`	Semelhante ao construtor anterior mas com o título da janela JDialog.

JDialog

Construtor	*Operação*
`JDialog(Frame f)`	Constrói uma nova instância de JDialog inicialmente invisível e vinculada a janela f.
`JDialog(Frame f, boolean m)`	Constrói uma nova instância de JDialog inicialmente invisível, definida como modal caso o parâmetro m seja verdadeiro.
`JDialog(Frame f, String titulo, boolean m)`	Semelhante ao construtor anterior mas com o título da janela JDialog.

JDialog

Método	*Uso*
`isModal()`	Retorna verdadeiro se o objeto JDialog questionado for modal.
`setModal(boolean b)`	Redefine o comportamento modal do objeto JDialog.

JDialog - exercício

Complementando o exercício CandidatoEmprego, crie uma classe em java (**DocumentosEntregues.java**) que seja subclasse de JDialog. Todas as características devem ser definidas no construtor da classe atual:

Frame-pai:	**janela** (da classe CandidatoEmprego)
Título:	Documentos Entregues
Modal:	true
Cor de Fundo:	(73,165,227)
Dimensão :	largura = 300 e altura = 470
Localização:	x = 200 e y = 200
Layout:	nulo

JDialog – exercício (interface final)

(complementar ao exercício CandidatoEmprego.java)

Imagens nos componentes: ícones

- O Swing inclui a implementação do ícone através da classe ImageIcon, que permite associar imagens a objetos JComponent. A ImageIcon possui vários construtores, seguem três deles:

```
public ImageIcon(String fileName)
public ImageIcon(URL url)
public ImageIcon(Image img)
```

- java pode manusear: GIF, JPEG e JPG.

JLabel

- Um área onde textos não editáveis e imagens podem ser mostrados
- Rótulos são usados para informações ou instruções textuais na interface gráfica
- Um rótulo imagem mostra texto não editável na posição selecionada
- Uma vez criado um rótulo, programas raramente modificam seu conteúdo

JLabel - construtores

```
public JLabel()
public JLabel (String s)
public JLabel (String s, int alignment)
public JLabel(String text,
              Icon icon,int horizontalAlignment)
public JLabel(Icon icon, int horizontalAlignment)
```

Constantes de alinhamento:

- `SwingConstants.LEFT`
- `SwingConstants.CENTER`
- `SwingConstants.RIGHT`

JLabel: Métodos

- public String getText()
  - retorna o texto do rótulo
- public void setText(String s)
  - coloca o texto s no rótulo
- public void setAlignment (int alignment)
  - alinha o texto do rótulo conforme as constantes de alinhamento vistas anteriormente.
- public void setVerticalTextPosition(int textPosition)
  - especifica a posição do texto com relação à imagem (TOP, CENTER (default), ou BOTTOM)
- public void getVerticalTextPosition()
- public void setHorizontalTextPosition(int textPosition)
  - especifica a posição do texto com relação à imagem (LEFT, CENTER, RIGHT)
- public void getHorizontalTextPosition()

JLabel - exemplo


```
import java.awt.*;
import javax.swing.*;
public class TesteLabel
{
   public static void main(String[] args)
   {
      JFrame fr = new JFrame();
      JLabel lb = new JLabel("Texto em baixo",
                             new ImageIcon("duke.gif"),
                             SwingConstants.LEFT);
      lb.setHorizontalTextPosition(SwingConstants.CENTER);
      lb.setVerticalTextPosition(SwingConstants.BOTTOM);
      fr.getContentPane().add(lb);
      fr.pack();
      fr.setVisible(true);
   }
}
```

JLabel - exercício

Utilize a classe CandidatoEmprego.java e lhe acrescente 5 atributos privados da classe JLabel. Esses atributos deverão ser instanciados no construtor da classe atual:

jlCabecalho : título “Ficha de Avaliação”, alinhamento CENTER, cor de frente azul, retângulo (130, 20, 250, 25) e fonte ("Arial", negrito, 18). *Adicionar ao ContentPane.*

jlNome : título “Nome”, alinhamento LEFT, cor de frente default, retângulo (15, 20, 40, 20) e fonte ("Arial", negrito, 12). *Adicionar ao painelIdentificacao.*

jlSexo: título “Sexo”, alinhamento LEFT, cor de frente default, retângulo (15, 45, 40, 20) e fonte ("Arial", negrito, 12). *Adicionar ao painelIdentificacao.*

jlInteresse: título “Interesse”, alinhamento LEFT, cor de frente default, retângulo (60, 10, 150, 20) e fonte ("Arial", negrito, 12). *Adicionar ao painelAreas.*

jlAtuacao: título “Atuação”, alinhamento LEFT, cor de frente default, retângulo (270, 10, 150, 20) e fonte ("Arial", negrito, 12). *Adicionar ao painelAreas.*

JTextComponent

- A classe JTextComponent é uma abstração de um componente que recebe inserções de textos do usuário.
- Ela é a super-classe abstrata de duas classes com funcionalidades em comum: JTextField e JTextArea e seus métodos e atributos são compartilhados por estas.

JTextComponent - métodos

- public String getText()
  - Retorna um objeto da classe String contendo o texto correspondente ao que o usuário digitou no objeto JTextComponent.
- public void setText(String texto)
  - Define o texto do objeto JTextComponent conforme o valor do objeto String texto passado como parâmetro.

JTextComponent - métodos

- public String getSelectedText()
  - Retorna um objeto da classe String contendo o texto correspondente ao que o usuário marcou no objeto JTextComponent.
- public void select(int inicio, int final)
  - Marca o texto do objeto JTextComponent conforme no intervalo correspondente ao valor de início e final passados como parâmetros.

JTextComponent - métodos

- public void selectAll()
  - Seleciona todo o texto no objeto TextComponent.
- public void setEditable(boolean estado)
  - Define o objeto JTextComponent como editável ou não.

JTextField

- Um objeto da classe JTextField é um campo texto na forma de uma linha na qual textos podem ser digitados pelo usuário através do teclado.
- A classe JTextField é uma sub-classe de JTextComponent, portanto pode-se aplicar nesta todos os métodos vistos anteriormente na classe JTextComponent.

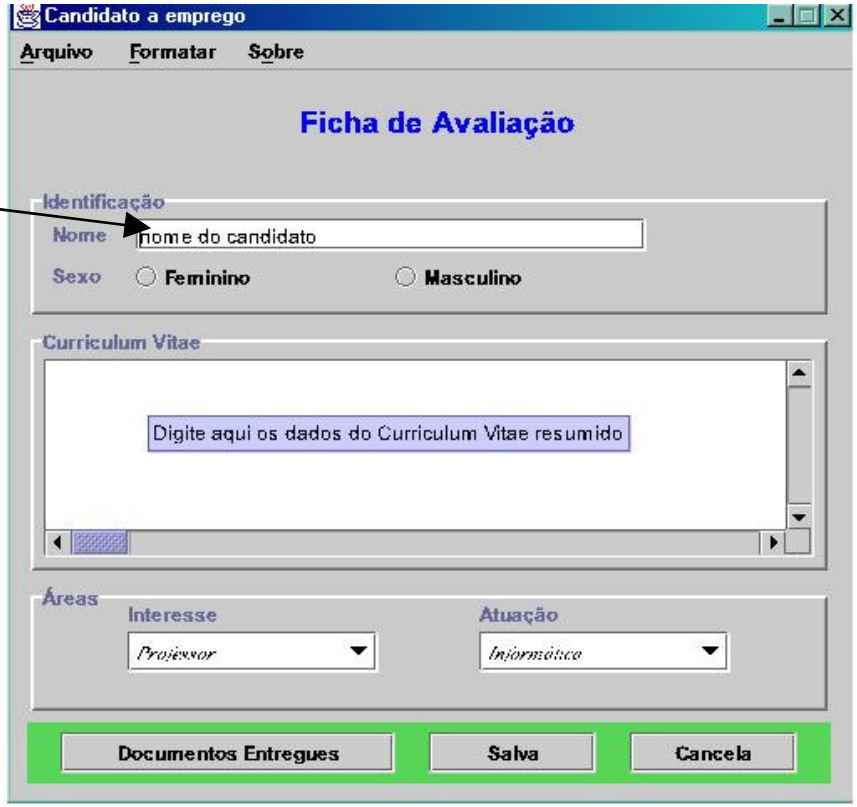
JTextField - construtores

- public JTextField()
  - constrói um objeto do tipo campo de texto.
- public JTextField(int colunas)
  - constrói um objeto campo de texto vazio com o número especificado de colunas.
- public JTextField(String texto)
  - constrói um objeto campo de texto com o texto fornecido no parâmetro texto.
- public JTextField(String s, int colunas)
  - constrói um objeto campo de texto com o texto fornecido no parâmetro texto com o número especificado de colunas.

JTextField - exercício

Utilize a classe CandidatoEmprego.java e lhe acrescente 1 atributo privado da classe JTextField. Esse atributo deverá ser instanciado no construtor da classe atual:

jtfNome: texto "nome do candidato", 70 colunas, posição (65, 20) e dimensão (305, 20) . *Adicionar ao painelIdentificacao.*



JTextArea

- Um objeto da classe JTextArea é um campo texto na forma de várias linhas nas quais textos podem ser digitados pelo usuário através do teclado.
- A classe JTextArea é também uma sub-classe de JTextComponent, portanto pode-se aplicar nesta todos os métodos vistos anteriormente na classe JTextComponent.

JTextArea - construtores

`public JTextArea()`

- constrói um objeto do tipo campo de texto com possibilidade para várias linhas.

`public JTextArea(int linhas, int colunas)`

- constrói um objeto JTextArea vazio com o número especificado de colunas e linhas.

`public JTextArea(String texto)`

- constrói um objeto JTextArea com o texto fornecido no parâmetro texto.

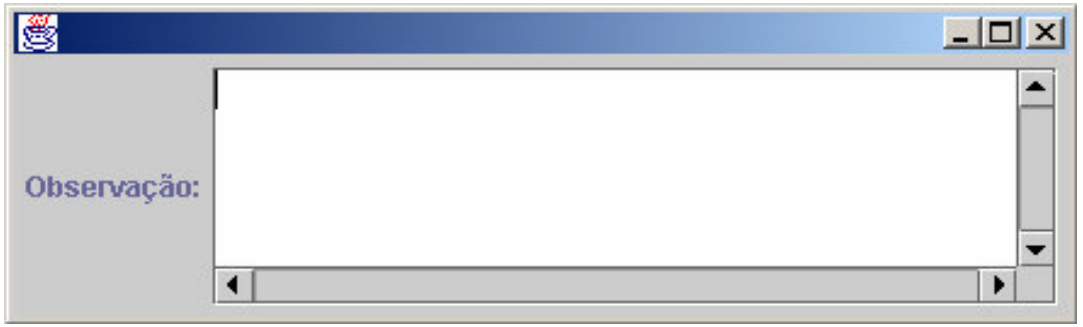
`public JTextArea(String texto , int linhas, int col)`

- constrói um objeto JTextArea com o texto especificado e número especificado de colunas e linhas.

JTextArea - métodos

- `public void setColumns(int c)`
  - define o número de colunas visíveis para o objeto JTextArea.
- `public int getColumns()`
  - devolve o número de colunas visíveis já definidas para o objeto JTextArea.
- `public void setRows(int r)`
  - define o número de linhas visíveis para o objeto JTextArea.
- `public int getRows()`
  - devolve o número de linhas visíveis já definidas para o objeto JTextArea.
- `append(String str)`
  - adiciona o texto no final do JTextArea.
- `insert(String str, int pos)`
  - insere o texto na posição especificada.

JTextArea - exemplo

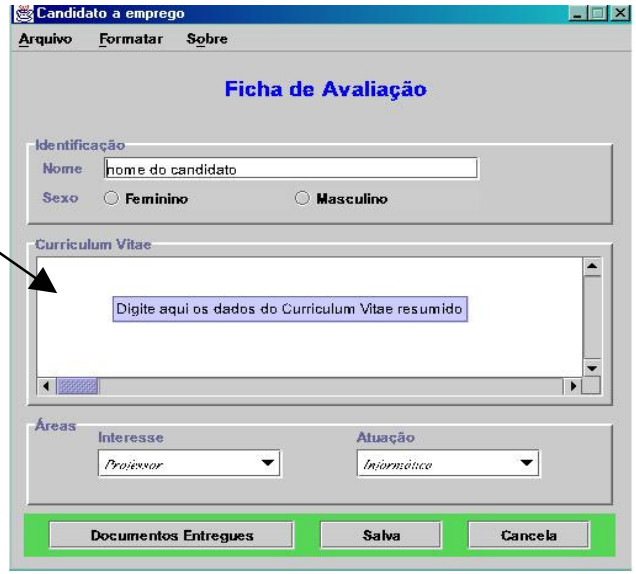



```
import java.awt.*;
import javax.swing.*;
public class TextAreaDemo
{
   public TextAreaDemo()
   {
      JLabel lNome = new JLabel("Observação:");
      JTextArea tArea = new JTextArea("",5,30);
      JScrollPane sp = new JScrollPane(tArea,
                           JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                            JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
      JPanel painelLabel = new JPanel();
      JFrame janelaDemo = new JFrame();
      painelLabel.add(lNome );
      painelLabel.add(sp );
      janelaDemo.getContentPane().add(painelLabel);
      janelaDemo.pack();
      janelaDemo.setVisible(true);
   }
   public static void main(String[] args)
   {
      TextAreaDemo t = new TextAreaDemo();
   }
}
```

JTextArea - exercício

Utilize a classe CandidatoEmprego.java e lhe acrescente 1 atributo privado da classe JTextArea. Esse atributo deverá ser instanciado no construtor da classe atual:

jtaCurriculum: texto " ", 6 linhas, 460 colunas e ToolTipText("Digite aqui os dados do Curriculum Vitae resumido"). NÃO ad*icionar a nenhum painel.*

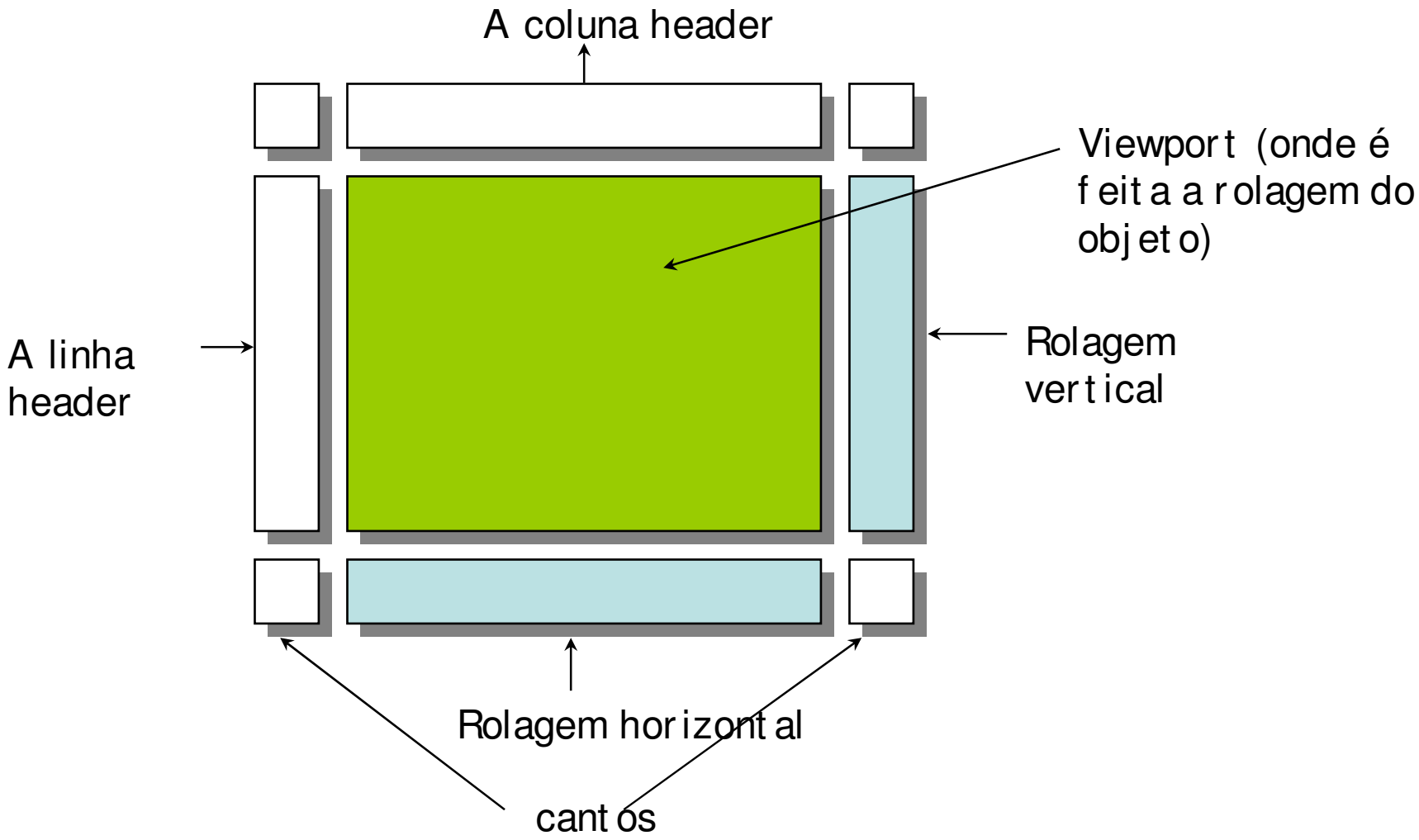


Adicione à classe CandidatoEmprego dois métodos públicos para definição e leitura dos valores digitados no objeto TextArea:
public void setCurriculum(String curriculum);
public String getCurriculum().

JScrollPane

- As mais sofisticadas interfaces de usuários utilizam áreas com barras de rolagem.
- O Swing inclui um conjunto de componentes em uma classe JScrollPane que atende estas demandas.
- Caso as características fornecidas pela classe JScrollPane não sejam suficientes, será possível criar novos componentes através da interação direta com os componentes da classe JScrollBar.
- Implementa as interfaces: ScrollPaneConstants e Accessible.

A estrutura de JScrollPane



JScrollPane

Construtor

`JScrollPane()` cria um JScrollPane vazio (nenhuma viewport) onde as áreas de scrollbar horizontal e vertical aparecem quando necessário.

`JScrollPane (Component view)`

cria um JScrollPane que exibe o conteúdo de um componente específico, onde os scrollbars horizontal e vertical aparecem quando o conteúdo do componente é maior do que o campo de visão.

JScrollPane

Construtor

`JScrollPane (Component view, int vsbPolicy, int hsbPolicy)`

cria um JScrollPane que exibe um campo de visão em uma viewport cuja posição do campo de visão pode ser controlada de acordo com as constantes da scrollbar.

`JScrollPane (int vsbPolicy, int hsbPolicy)`

cria um JScrollPane vazio (sem viewport) de acordo com constantes scrollbar especificadas.

JScrollPane

Constantes

VERTICAL_SCROLLBAR_AS_NEEDED

VERTICAL_SCROLLBAR_NEVER

VERTICAL_SCROLLBAR_ALWAYS

HORIZONTAL_SCROLLBAR_AS_NEEDED

HORIZONTAL_SCROLLBAR_NEVER

HORIZONTAL_SCROLLBAR_ALWAYS

JScrollPane

Métodos

`setHorizontalscrollBarPolicy (int policy)`

determina quando a scrollbar horizontal aparecerá na scrollpane.

`setVerticalscrollBarPolicy (int policy)`

determina quando a scrollbar veritcal aparecerá na scrollpane.

`setViewportView(Component view)`

Creates a viewport if necessary and then sets its view.

JScrollPane - exercício

Utilize a classe CandidatoEmprego.java e lhe acrescente 1 atributo privado da classe JScrollPane. Esse atributo deverá ser instanciado no construtor da classe atual com os parâmetros:

jspCurriculum: (jtaCurriculum, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS, JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS), posição (10, 20) e dimensão (460, 120) .
Adicionar ao painelCurriculum.

Botões

- O pacote javax.swing suporta os seguintes tipos de botões:

• **JButton**	Botão comum
• **JCheckBox**	Botões de seleção independentes
• **JRadioButton**	Botões agrupados para seleção
• **JMenuItem**	Item de menu no formato de botão JCheckBox ou JRadioButton.

JButton

É subclasse de AbstractButton e Jcomponent.

- **public JButton( )**
  - constrói um objeto do tipo Botão sem texto.
- **public JButton(String texto)**
  - constrói um objeto Botão com o texto especificado.
- **public JButton(Icon icon)**
  - constrói um objeto Botão com o ícone especificado.
- **public JButton(String texto, Icon icon)**
  - constrói um objeto Botão com o texto especificado.

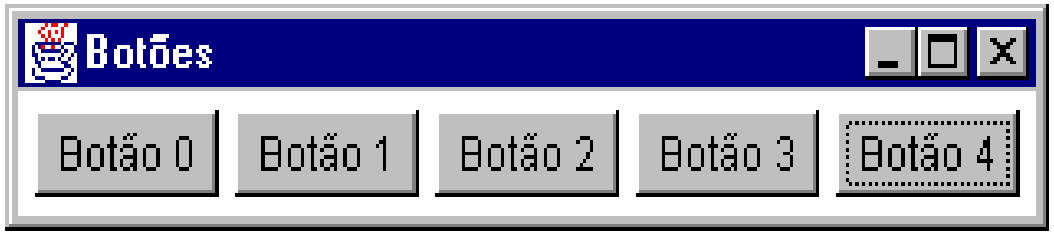
JButton - métodos

- **`getText()`**
  - retorna o String representando o texto vinculado ao botão.
- **`setText(String s)`**
  - define um texto para representar o botão.
- **`getIcon()`**
  - retorna o ícone vinculado ao botão.
- **`setIcon(Icon icon)`**
  - define um ícone para ser mostrado pelo botão.

JButton - métodos

- **`setDisabledIcon(Icon icon)`**
  - atribui o ícone a ser exibido quando o botão for inibido.
- **`setPressedIcon(Icon icon)`**
  - atribui o ícone a ser exibido quando o botão for pressionado.
- **`setSelectedIcon(Icon icon)`**
  - atribui o ícone a ser exibido quando o botão for selecionado.
- **`setRolloverIcon(Icon icon)`**
  - atribui o ícone a ser exibido quando o botão for selecionado.

JButton – exemplo




```
import java.awt.*;
public class ButtonDemo
{
   private Button[] botoes;
   private Frame janelaDemo;
   public ButtonDemo()
   {
      janelaDemo = new Frame("Botões");
      janelaDemo.setLayout(new FlowLayout());
      botoes = new Button[5];
      for(int i = 0; i< botoes.length;i++)
      {
         botoes[i] = new Button("Botão "+i);
         janelaDemo.add(botoes[i]);
      }
      janelaDemo.pack();
   }
   public void mostraExemplo()
   {
      janelaDemo.setVisible(true);
   }
   public static void main(String[] args)
   {
      ButtonDemo  t = new ButtonDemo();
      t.mostraExemplo();
   }
}
```

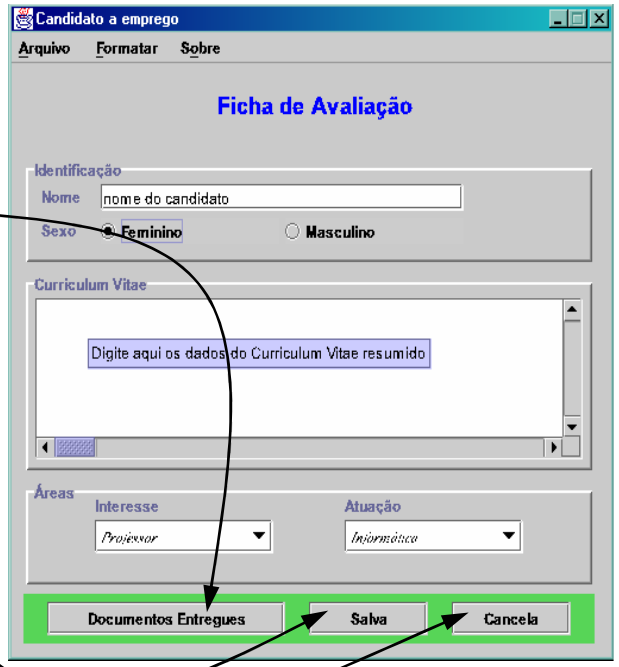
JButton - exercício

Utilize a classe CandidatoEmprego.java e lhe acrescente 3 atributos privados da classe JButton. Esses atributos deverão ser instanciados no construtor da classe atual com os parâmetros:

jbDocumentosEntregues:texto "Documentos Entregues", retângulo (20,6,200,25) e **ToolTipText**("Clique aqui para acessar a tela de Documentos Entregues"). *Adicionar ao painelBotoes.*

jbSalva: texto "Salva" e retângulo (240,6,100,25). *Adicionar ao painelBotoes.*

jbCancela: texto "Cancela" e retângulo (360,6,100,25). *Adicionar ao painelBotoes.*



JRadioButton

- É subclasse de JToggleButton, AbstractButton e Jcomponent.
- Pode ter dois estados : ligado e desligado.
- Podem ser agrupados pela classe *ButtonGroup.* Neste caso, a seleção é mutuamente exclusiva.

JRadioButton

```
import java.awt.*;
import javax.swing.*;
public class MeuRadioButton extends JFrame
{
   public static void main(String args[])
   {
      MeuRadioButton window = new MeuRadioButton ();
      JPanel p = new JPanel();
      window.getContentPane().add(p);
      JRadioButton masculino = new JRadioButton( "Masculino" );
      p.add( masculino );
      JRadioButton feminino = new JRadioButton( "Feminino" );
      p.add( feminino );
      ButtonGroup grupo = new ButtonGroup ();
      grupo.add(masculino);
      grupo.add(feminino);
      window.pack();
      window.setVisible(true);
 }
}
```

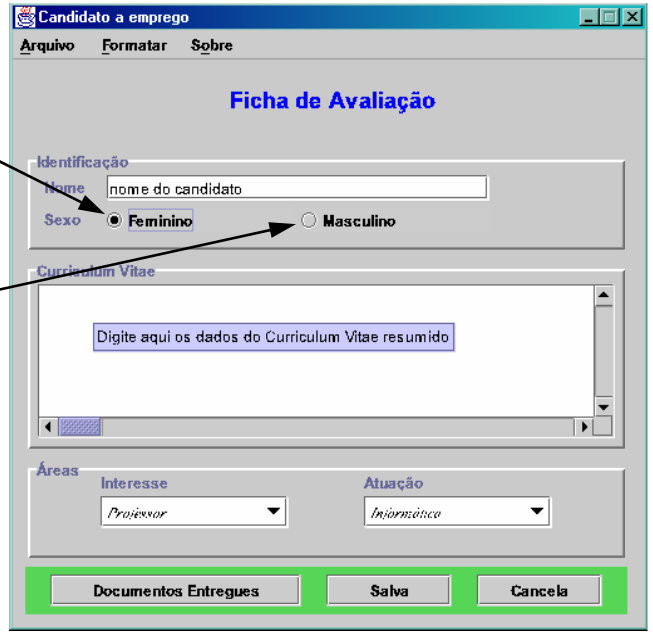
JRadioButton - exercício

Utilize a classe CandidatoEmprego.java e lhe acrescente 3 atributos privados: 2 da classe JRadioButton e outro da classe ButtonGroup . Esses atributos deverão ser instanciados no construtor da classe atual com os parâmetros:

jrbfeminino: texto "Feminino", retângulo(65,45,150,20) e selecionado = true. *Adicionar ao painelIdentificacao* e ao *bgGrupoSexo.*

jrbMasculino: texto "Masculino" e retângulo(220,45,150,20). *Adicionar ao painelIdentificacao* e ao *bgGrupoSexo.*

bgGrupoSexo: adicionar lhe os objetos jrbFeminino e jrbMasculino. NÃO ad*icionar a nenhum painel.*



JCheckBox

- É subclasse de JToggleButton, AbstractButton e Jcomponent.
- Igualmente ao *JRadioButton*, um *JCheckBox* pode ter dois estados: ligado ou desligado.
- Um conjunto de checkboxes podem ser agrupados através da classe ButtonGroup. Neste caso, a seleção é mutuamente exclusiva.

JCheckBox

Construtores	Uso
`JCheckBox(String label)`	Cria um Checkbox desligado, com o label especificado.
`JCheckBox(String label,boolean state)`	Cria um Checkbox ligado ou desligado, de acordo com a variável booleana state e com o label especificado.
`JCheckBox(Icon icon, boolean state)`	Cria um Checkbox ligado ou desligado, de acordo com a variável booleana state, com o ícone especificado.

JCheckBox - exemplo

```
import java.awt.*;
import javax.swing.*;
public class MeuCheckbox extends JFrame
{
   public static void main(String args[])
   {
      MeuCheckbox window = new MeuCheckbox();
      JPanel p = new JPanel();
      window.getContentPane().add(p);
      JCheckBox negrito = new JCheckBox( "Negrito" );
      p.add( negrito );
      JCheckBox italico = new JCheckBox( "Itálico" );
      p.add( italico );
      window.pack();
      window.setVisible(true);
 }
}
```

JCheckBox - exercício

Utilize a classe DocumentosEntregues.java e lhe acrescente 5 atributos privados da classe JCheckBox (não agrupados). Esses atributos deverão ser instanciados no construtor da classe atual e adicionados ao **ContentPane** correspondente:

jcbDoc1: texto "Carteira de Identidade e CPF", ativo = true, cor de fundo (255,204,0), cor de frente preta, retângulo(10,20,270,45).

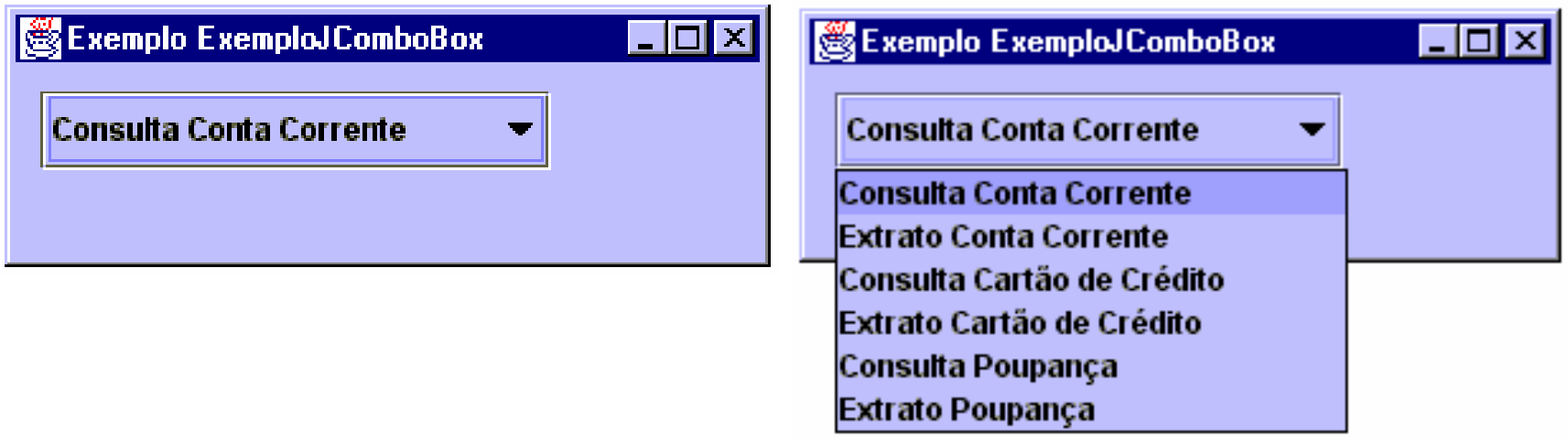
jcbDoc2: texto "Carteira de Trabalho", ativo = false, cor de fundo default, cor de frente preta, retângulo(10,65,270,45).

jcbDoc3: texto "Curriculum resumido e Comprovantes", ativo = false, cor de fundo (255,204,0), cor de frente preta, retângulo(10,110,270,45).

jcbDoc4: texto "Foto 3x4", ativo = true, cor de fundo default, cor de frente preta, retângulo(10,155,270,45).

jcbDoc5: texto "Proposta de Trabalho", ativo = true, cor de fundo (255,204,0), cor de frente preta, retângulo(10,200,270,45).

JComboBox



- É herança direta de **Jcomponent**.
- Um JComboBox é um componente visual que possibilita a manipulação de coleções de objetos permitindo ao usuário selecionar apenas um objeto da coleção. Cada objeto inserido no JComboBox é representado visualmente pela String retornada pelo método toString().

JComboBox

- Um objeto JComboBox utiliza o MVC (Model Viewer Controler), possibilitando ao desenvolvedor criar sua própria massa de dados através de classes de manipulação fornecidas no pacote Swing.
- Um objeto JComboBox pode também ser manipulado diretamente, recebendo elementos para montagem de sua massa de dados um a um através do método addItem.
- objetos da classe JComboBox assumem também comportamento semelhante ao componente JTextField, tornando possível editar os componentes selecionáveis e pesquisar rapidamente um elemento digitando a primeira letra dele.

JComboBox

Construtores

- **public JComboBox(ComboBoxModel aModel)**
  - Cria um objeto JComboBox que recebe seus itens de um ComboBoxModel previamente definido.
- **public JComboBox(Object[] items)**
  - Cria um objeto JComboBox que recebe seus itens de um array de objetos. O objeto JComboBox utilizará o método toString de cada objeto do array.
- **public JComboBox(Vector items)**
  - Cria um objeto JComboBox que recebe seus itens de um vetor de objetos. O objeto JComboBox utilizará o método toString de cada objeto do vetor.
- **public JComboBox()**
  - Cria um objeto JComboBox com um modelo de dados padrão (uma lista de objetos vazia).

JComboBox

Métodos

- **public void setModel(ComboBoxModel aModel)**
  - Redefine o modelo de dados que o JComboBox utilizará.
- **public ComboBoxModel getModel()**
  - Resgata o modelo de dados que o JComboBox está utilizando.
- **public void setEditable(boolean aFlag)**
  - Faz com que o JCombobox se comporte como um JTextField, possibilitando a edição ou seleção dos elementos cadastrados. (a edição não afeta a massa de dados, apenas o campo)
- **public boolean isEditable()**
  - Informa se o objeto JComboBox está editável ou não.
- **public void setMaximumRowCount(int count)**
  - Define o número máximo de linhas que o JComboBox irá mostrar.
- **public int getMaximumRowCount()**
  - Informa o máximo de linhas que o JComboBox poderá mostrar.

JComboBox

Métodos

- **public void setSelectedItem(Object anObject)**
  - Seleciona um determinado objeto da lista no JComboBox.
- **public Object getSelectedItem()**
  - Retorna o item correntemente selecionado.
- **public void setSelectedIndex(int anIndex)**
  - Seleciona um determinado objeto da lista no JComboBox através do seu valor posicional.
- **public int getSelectedIndex()**
  - Retorna o posicional do item correntemente selecionado.
- **public void addItem(Object anObject)**
  - Adiciona um elemento na lista padrão (usado quando o JComboBox é criado com construtor vazio).

JComboBox

Métodos

- **public void insertItemAt(Object anObject, int index)**
  - Insere um elemento na lista padrão na posição informada pelo índice (usado quando o JComboBox é criado com construtor vazio).
- **public void removeItem(Object anObject)**
  - Remove o elemento na lista padrão (usado quando o JComboBox é criado com construtor vazio).
- **public void removeItemAt(int anIndex)**
  - Remove o elemento do índice informado na lista padrão (usado quando o JComboBox é criado com construtor vazio).
- **public void removeAllItems()**
  - Remove todos elementos da lista padrão (usado quando o JComboBox é criado com construtor vazio).

JComboBox

Métodos

- **public void setEnabled(boolean b)**
  - Define se o objeto poderá ser manipulado pelo usuário.
- **public int getItemCount()**
  - Retorna o número de elementos no objeto JComboBox.
- **public Object getItemAt(int index)**
  - Retorna o objeto referente ao elemento posicionado no índice fornecido.

JComboBox - exemplo

```
import javax.swing.*;
import java.awt.*;
public class ExemploJComboBox {
public ExemploJComboBox() {
        JFrame f = new JFrame("Exemplo ExemploJComboBox");
        JComboBox comboBox = new JComboBox();
        comboBox.addItem("Consulta Conta Corrente");
        comboBox.addItem("Extrato Conta Corrente");
        comboBox.addItem("Consulta Cartão de Crédito");
        comboBox.setBounds(10,10,200,30);
        f.getContentPane().setLayout(null);
        f.getContentPane().add(comboBox);
        f.setSize(300,100);
        f.setVisible(true);
    }
    public static void main(String args[]) {
        ExemploJComboBox e = new ExemploJComboBox();
    }
}
```

JComboBox – exemplo com array

```
import javax.swing.*;
import java.awt.*;
public class ExemploJComboBoxArray {
   public ExemploJComboBoxArray() {
       JFrame f = new JFrame("Exemplo com Array");
       String[] dados = {"Melancia", "Abacate", "Melão" };
       JComboBox comboBox = new JComboBox(dados);
       comboBox.setBounds(10,10,200,30);
       f.getContentPane().setLayout(null);
       f.getContentPane().add(comboBox);
       f.setSize(300,100);
       f.setVisible(true);
   }
   public static void main(String args[]) {
      ExemploJComboBoxArray e = new ExemploJComboBoxArray();
   }
}
```

JComboBox - **ComboBoxModel**

- Conforme demonstrado nos exemplos anteriores, um objeto JComboBox pode ser utilizado de forma semelhante ao objeto Choice do pacote AWT (recebendo elementos um a um) ou recebendo uma coleção de objetos via vetor ou array de objetos. Outra forma de se utilizar um objeto JComboBox é através da criação de um modelo de dados que se associa à classe JComboBox via método construtor - **public JComboBox(ComboBoxModel aModel)** - ou por uma chamada ao método:

  public void setModel(ComboBoxModel aModel).

- O modelo de dados **ComboBoxModel** é uma interface e subclasse de outra interface (**ListModel**) que contém métodos pré-definidos para manipulação e consulta de uma massa de dados por um JComboBox**.**

JComboBox - **ComboBoxModel**

Métodos

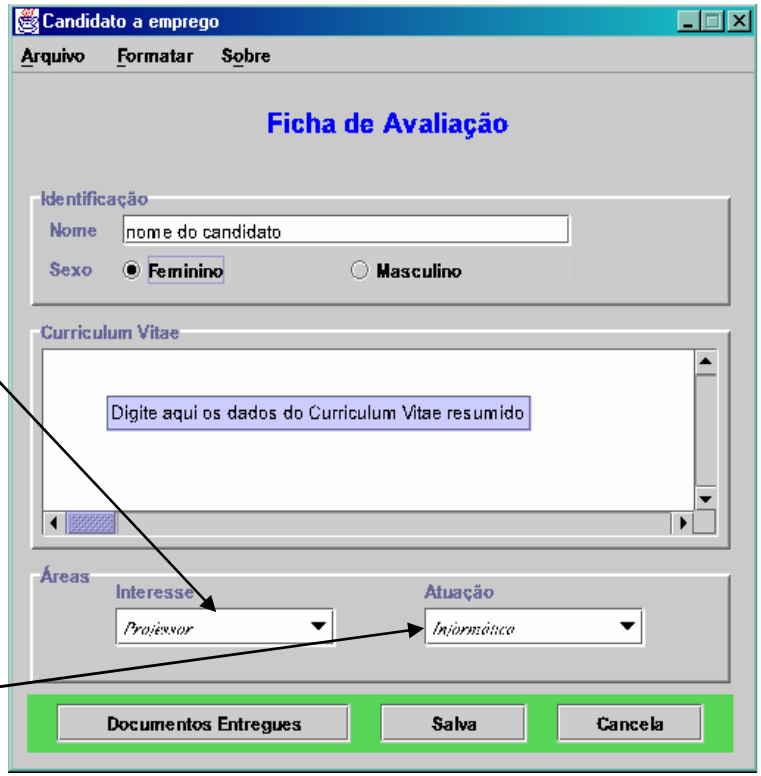
- **public int getSize()**
  - Deverá ser implementado para retornar o número de elementos da coleção de dados.
- **public Object getElementAt(int index)**
  - Deverá retornar o objeto na posição informada pelo índice.
- **public void setSelectedItem(Object anItem)**
  - Deverá marcar o elemento como selecionado.
- **public Object getSelectedItem()**
  - Deverá devolver o elemento que foi previamente selecionado.

JComboBox - exercício

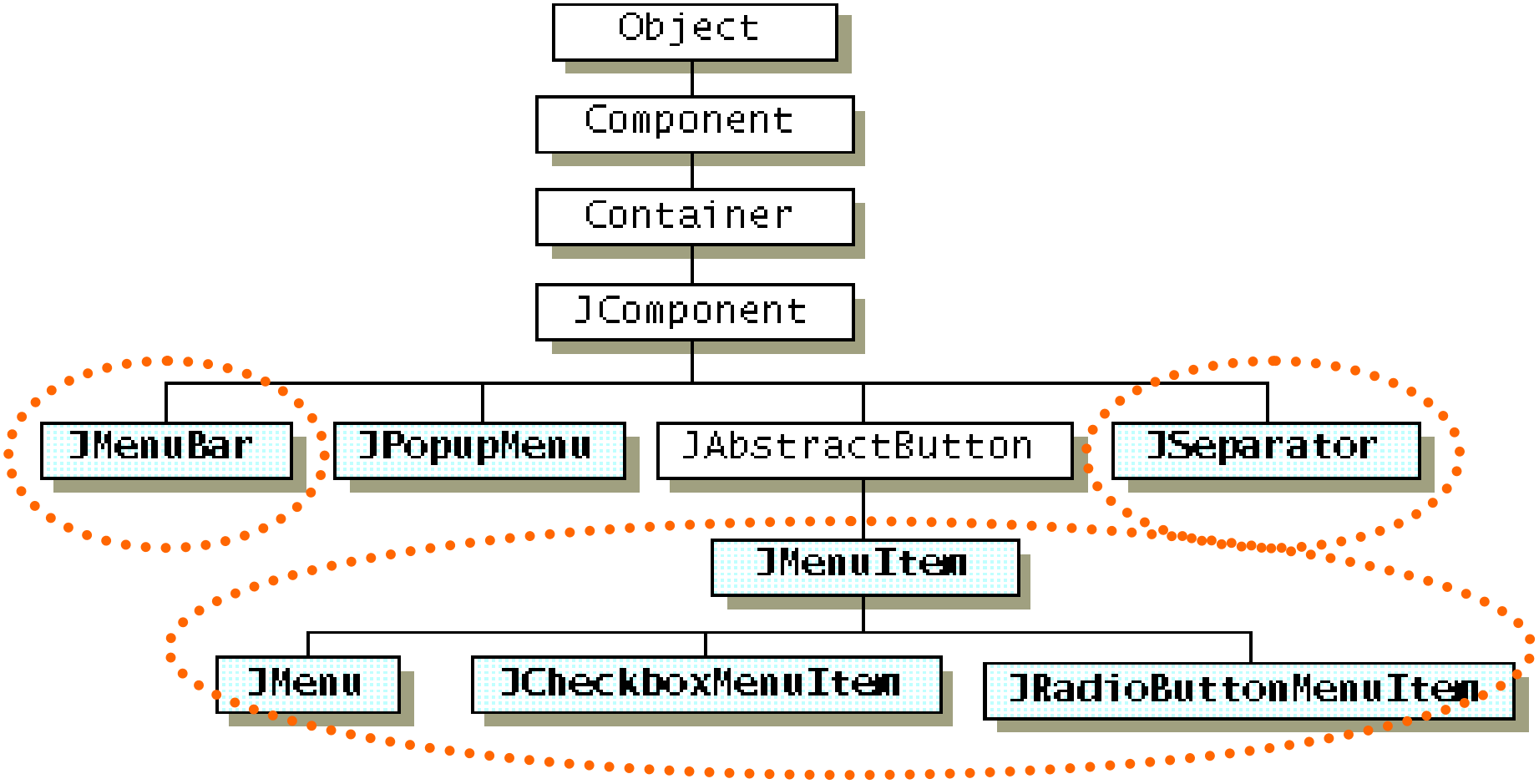
Utilize a classe CandidatoEmprego.java e lhe acrescente 2 atributos privados da classe JComboBox. Esses atributos deverão ser instanciados no construtor da classe atual com os parâmetros:

jcbInteresse: cor de fundo branca, retângulo(60,30,150,25), fonte ("TimesRoman",Font.ITALIC,12) e ToolTipText("Escolha a Área de Interesse"). Adicionar as Strings ("Professor", "Gerente", "Desenvolvedor" e "Outra"). *Adicionar ao painelAreas.*

jcbAtuacao: cor de fundo branca, retângulo (270,30,150,25) e fonte ("TimesRoman",Font.ITALIC,12). Adicionar as Strings ("Informática", "Matemática", "Biologia" e "Outra"). *Adicionar ao painelAreas.*



Criando Menus *LFSC*



Fonte : http://java.sun.com/docs/books/tutorial/uiswing/components/menu.html

JMenuBar *LFSC*

É subclasse de JComponent. O layout default é o BoxLayout.

`JMenuBar( )`	Constrói um objeto do tipo barra de menu.
`add(JMenu c)`	Adiciona o menu especificado ao final da barra de menu.

Para atribuir a barra de menu ao JFrame desejado, basta utilizar o método da janela:

```
void setJMenuBar(JMenuBar menubar)
```

JMenuItem *LFSC*

É subclasse de JAbstractButton, ou seja, os itens de menu são simplesmente botões.

Construtores

JMenuItem( )	Cria um item de menu.
JMenuItem(String s)	Cria um item de menu com o texto especificado.
JMenuItem(String s, Icon i)	Cria um item de menu com o texto e ícone especificados.
JMenuItem(String s, int mnemonic)	Cria um item de menu com o texto e mnemônico especificados.

JMenuItem *LFSC*

Alguns métodos importantes :

•void **setMnemonic(int mnemônico)**

-- Herdado de JAbstractButton, atribui uma tecla de acesso ao item de menu.

ex.: itemDeMenu.**setMnemonic**(72); // char N.

•void **setAccelerator(KeyStroke keyStroke)**

-- Atribui uma tecla de atalho para execução direta do item de menu.

ex.: itemDeMenu.**setAccelerator**(KeyStroke.getKeyStroke(KeyEvent.VK_B, ActionEvent.ALT_MASK));

***resultará na tecla de atalho ALT_B.**

JMenu *LFSC*

É subclasse de JMenuItem. Utilizada para criar os itens principais de menu ou submenus.

JMenu( )	Cria um menu principal ou submenu. Para atribuir um ícone, utilize **setText(String s)**.
JMenu(String s)	Cria um menu principal ou submenu com o texto especificado. Para atribuir um ícone, utilize **setIcon(Icon i)**.
add(String s)	Cria um item de menu com o texto especificado e o adiciona ao final do menu.
add(JMenuItem j)	Adiciona o item de menu especificado ao final da barra de menu.
addSeparator( )	Adiciona um separador como próximo (ou último) item do menu.

Como criar um menu *LFSC*

1. Criar primeiramente a JMenuBar( ) e atribuí-la ao JFrame desejado.

2. Criar os menus principais ou submenus do tipo JMenu( ) e adicioná-los à barra de menu (JMenuBar) ou a um menu (Jmenu) como um novo item.

3. Criar os itens de menu do tipo JMenuItem( ) e adicioná-los ao menu principal ou ao submenu correspondente (JMenu).

4. Se houver subitens de menu, repetir os passos 2 e 3.

JMenuBar e JMenu - exercício *LFSC*

Componentes a serem implementados na classe **CandidatoEmprego.java** :

- Criar uma barra de menu (**jmbBarraMenu**) e atribuí-la à **janela**.
- Criar 3 menus principais e adicioná-los à "**barra de menu**" :

- **jmArquivo**: texto "Arquivo", mnemônico 'A' (int 65)' e ícone "tips.gif".
- **jmFormatar**: texto "Formatar", mnemônico 'F' (int 70).
- **jmSobre**: texto "Sobre", mnemônico 'o' (int 111).

JMenuItem - exercício 1 *LFSC*

Componentes a serem implementados na classe **CandidatoEmprego.java** :

- Criar 5 itens de menu e adicioná los ao menu "**Arquivo**" :

- **jmiNovo**: texto "Novo" (20 caracteres de largura), mnemônico 'N' (int 78) e tecla aceleradora "ALT_B".
- **jmiAbrir**: texto "Abrir".
- **jmiFechar**: texto "Fechar".
- **jmiSalvar**: texto "Salvar".
- **jmiSair**: texto "Sair".

*Incluir a instrução **import java.awt.event.*** .

JMenuItem - exercício 2 *LFSC*

Componentes a serem implementados na classe **CandidatoEmprego.java** :

- Criar 2 submenus e adicioná-los ao menu "**Formatar**" :
  - **jmCor**: texto "Cor".
  - Adicionar um separador (addSeparator( )).
  - **jmFonte**: texto "Fonte".

- Criar 4 itens de menu do tipo JRadioButtonMenuItem (agrupados) e adicioná-los ao submenu **jmCor** :
  - **jrbmiAzul**: texto "Azul" e selecionado = true.
  - **jrbmiVerde**: texto "Verde".
  - **jrbmiVermelho**: texto "Vermelho".
  - **jrbmiPreto**: texto "Preto".

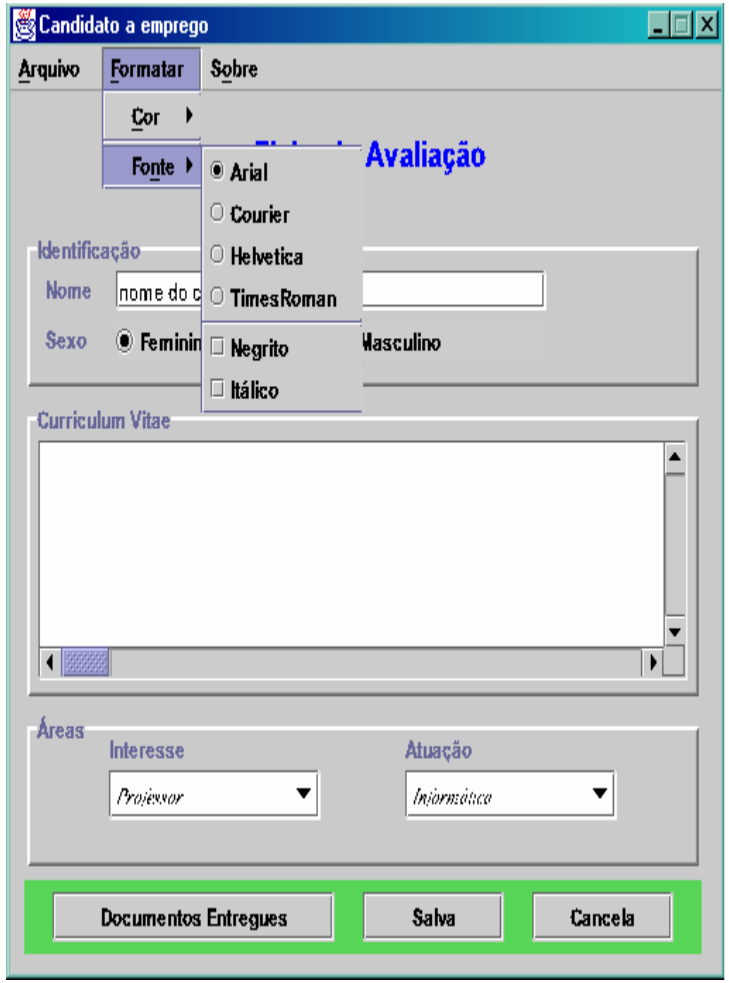
*Adicioná-los também ao ButtonGroup correspondente.

JMenuItem - exercício 3 *LFSC*

Componentes a serem implementados na classe **CandidatoEmprego.java** :

- Criar 4 itens de menu do tipo JRadioButtonMenuItem (agrupados) e adicioná-los ao submenu **jmFonte** :
  - **jrbmiArial**: texto “Arial” e selecionado = true.
  - **jrbmiCourier**: texto “Courier”.
  - **jrbmiHelvetica**: texto “Helvetica”.
  - **jrbmiTimesRoman**: texto “TimesRoman”.
  - Adicionar um separador (addSeparator( )).

  *Adicioná-los também ao **ButtonGroup** correspondente.



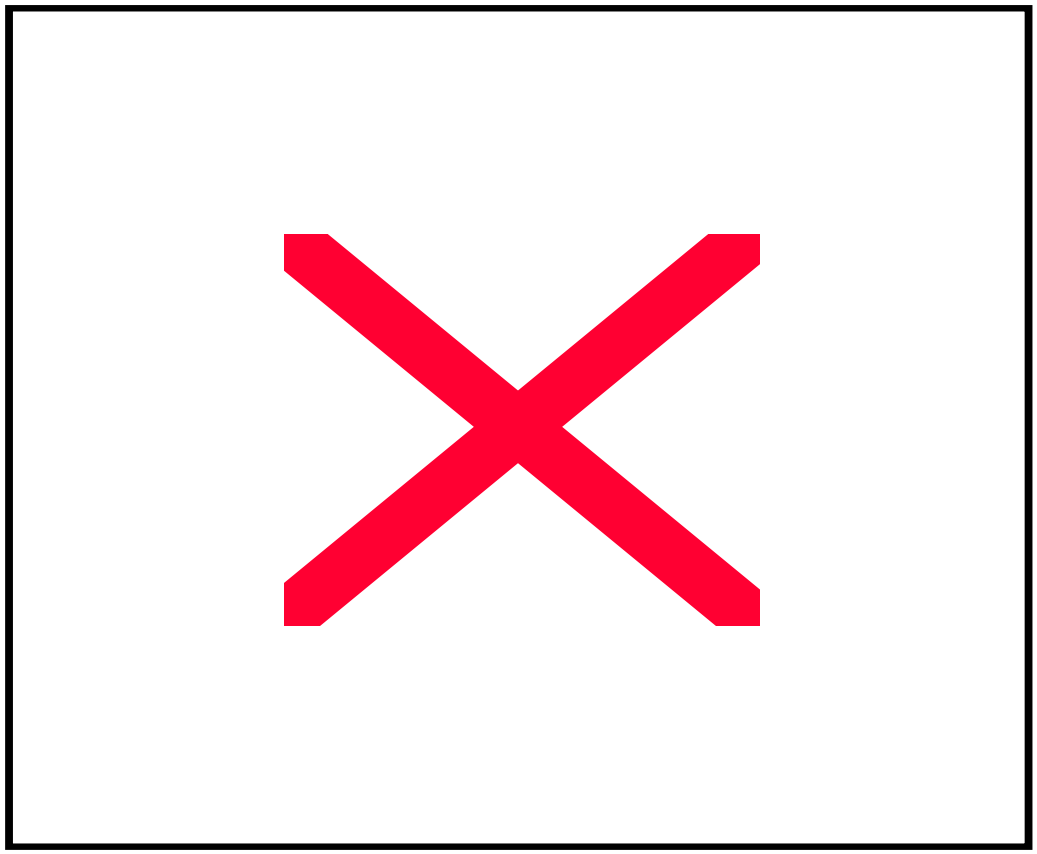
- Criar 2 itens de menu do tipo JCheckBoxMenuItem e adicioná-los ao submenu **jmFonte** :
  - **jcbmiNegrito**: texto “Negrito”.
  - **jcbmiItalico**: texto “Itálico” e selecionado = true.

JMenuItem - exercício 4 *LFSC*

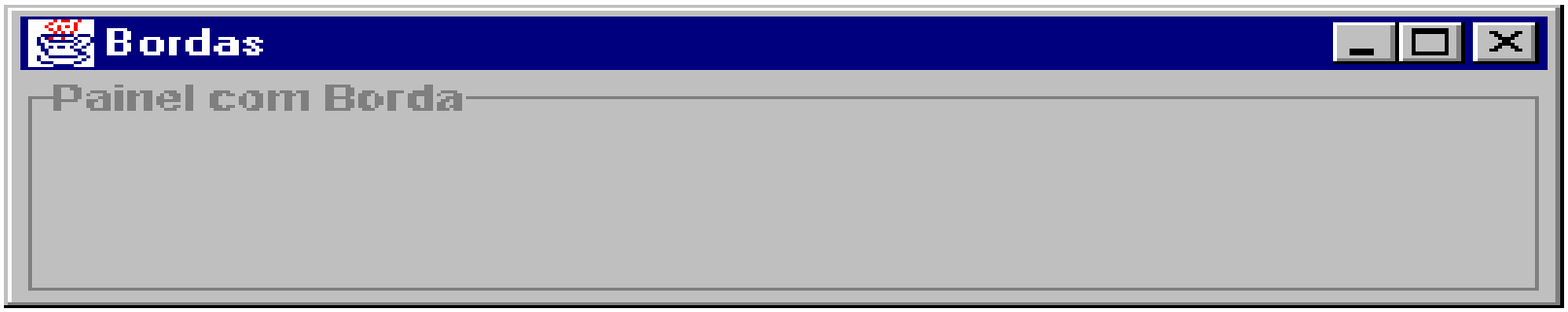
Componentes a serem implementados na classe **CandidatoEmprego.java** :

- Criar 1 item de menu e adicioná-lo ao menu "**Sobre**" :

  - **jmiSobre**: texto "Sobre a Aplicação" e ícone "tips.gif".



Border



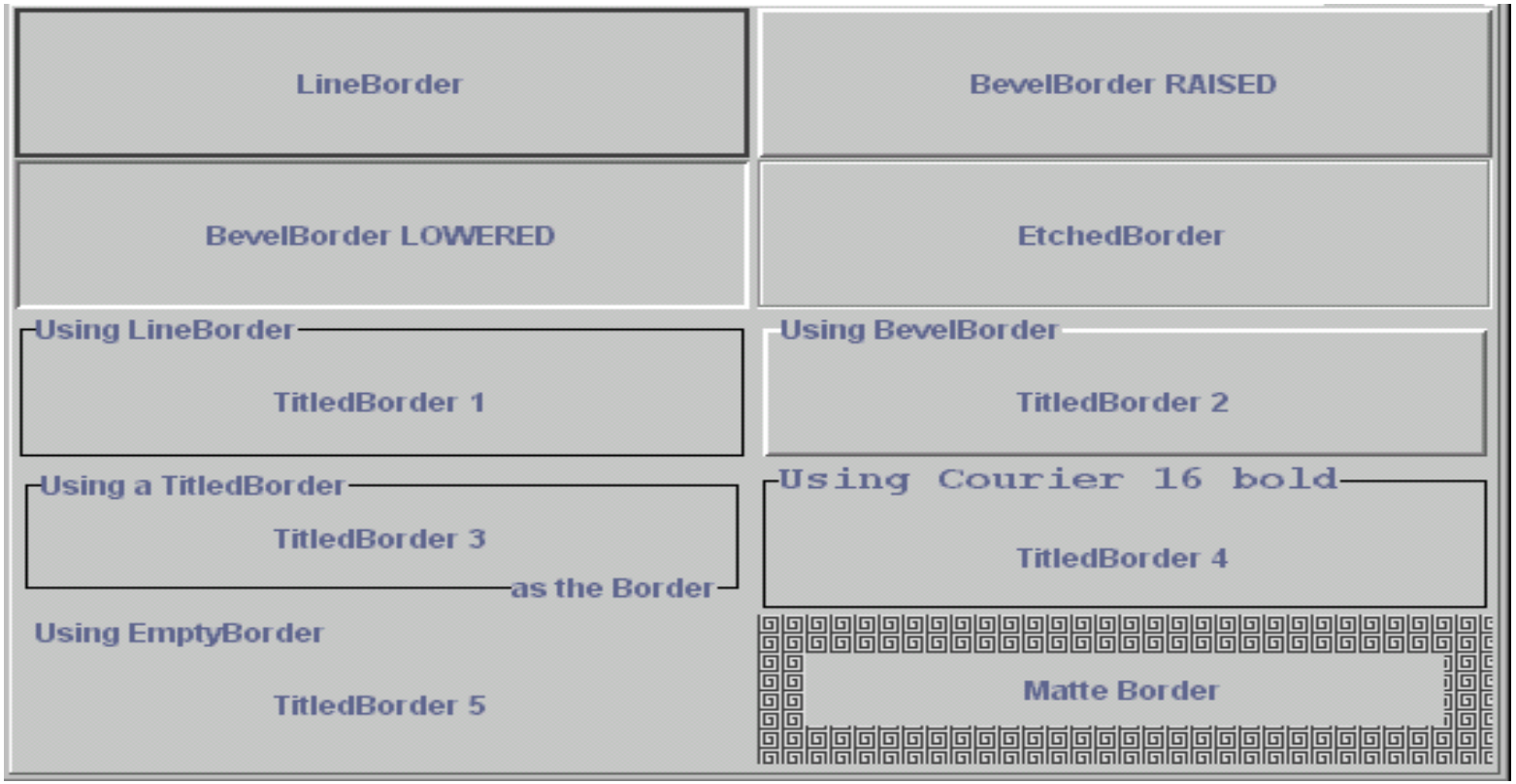
- A **interface Border** (javax.swing.border.*) permite a criação de vários modelos de bordas.
- A classe JComponent fornece o método public void setBorder(Border border), que permite a configuração da borda em componentes.
- O Swing também fornece uma classe utilitária de nome BorderFactory, que se encarrega de criar instâncias de bordas pré-definidas.

*Incluir a instrução **import javax.swing.border.***.

Border

Tipos pré-definidos de bordas:

BevelBorder, CompoundBorder, EmptyBorder, EtchedBorder, LineBorder, LoweredBevelBorder, MatteBorder, RaisedBevelBorder e TitledBorder.



BorderFactory

- `static Border createEmptyBorder()`
- `static Border createEtchedBorder()`
- `static Border createLineBorder(Color color)`
- `static Border createLoweredBevelBorder()`
- `static Border createRaisedBevelBorder()`
- `static TitledBorder createTitledBorder(Border border)`
- `static TitledBorder createTitledBorder(Border border, String title)`
- `static CompoundBorder createCompoundBorder( Border outsideBorder, Border insideBorder)`

Etapas na utilização de bordas

1. **Criar o objeto visual que conterá a borda:**

```
JPanel painelComBorda      = new JPanel();
```

2. **Criar o(s) objeto(s) bordas por meio da classe BorderFactory . Segue o exemplo de uma TitledBorder com uma RaisedBevelBorder:**

```
Border rbbBorda =
        BorderFactory.createRaisedBevelBorder();
TitledBorder tbBorda =  BorderFactory.createTitledBorder
        ( rbbBorda, "Painel com TitledBorder" );
```

3. **Caso existam ajustes de personalização na borda, efetuá-los antes de atribuí-la ao objeto visual. Os parâmetros abaixo se referem a uma TitledBorder:**

```
tbBorda.setTitlePosition(TitledBorder.TOP);
tbBorda.setTitleJustification(TitledBorder.LEFT);
```

4. **Vincular a borda ao objeto visual:**

```
painelComBorda.setBorder(tbBorda);
```

Border - exercício 1 *LFSC*

Componentes a serem implementados na classe **CandidatoEmprego.java** :

- Criar 5 objetos do tipo Border :
  - **bbBorda**: borda BevelBorder, tipo BevelBorder.RAISED, destaque verde e sombra laranja.
  - **rbbBorda**: borda RaisedBevelBorder.
  - **ebBorda**: borda EtchedBorder, destaque cinza escuro e sombra branca.
  - **lbBorda**: borda LineBorder, cor (3, 255, 3) e espessura 3.
  - **lbbBorda**: borda LoweredBevelBorder.

- Criar 3 objetos do tipo Titled**d**Border, com os seguintes parâmetros:
  - **tbIdentificacao**: borda **bbBorda** e título “Identificação”.
  - **tbCurriculum**: borda **rbbBorda** e título “Curriculum Vitae”.
  - **tbAreas** : borda **ebBorda** e título “Áreas”.

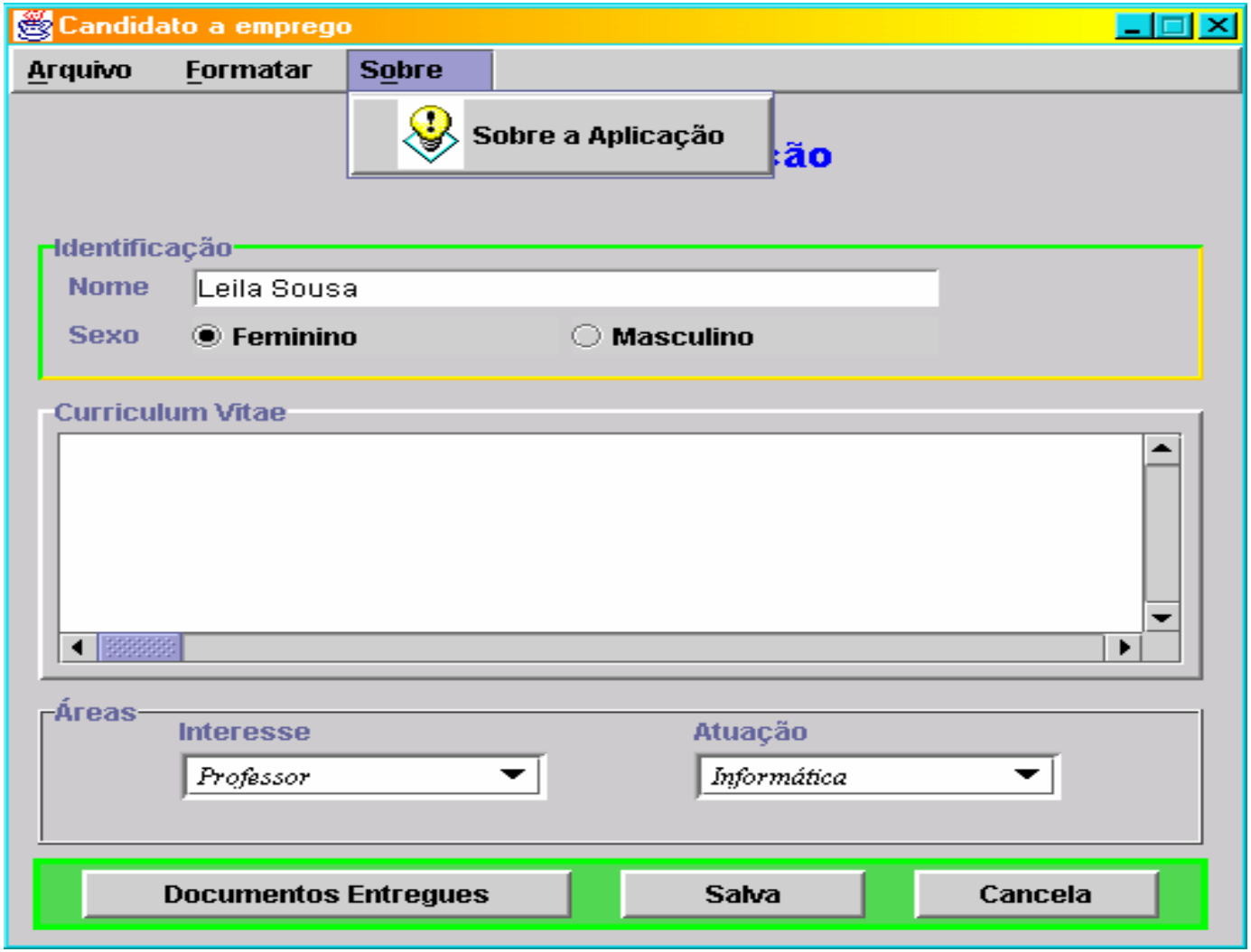
Border - exercício 2 *LFSC*

Componentes a serem implementados na classe **CandidatoEmprego.java** :

- Atribuir as bordas aos seguintes componentes:
  - borda **rbbBorda** aos jmbBarraMenu, jmiSobre, jbDocumentosEntregues, jbSalva e jbCancela.
  - borda **tbIdentificacao** ao painelIdentificacao.
  - borda **tbCurriculum** ao painelCurriculum.
  - borda **tbAreas** ao painelAreas.
  - borda **lbBorda** ao painelBotoes
  - borda **lbbBorda** ao jtfNome.
  - borda **lbbBorda** aos jcbInteresse e jcbAtuacao.

Border - resultado final *LFSC*

Componentes a serem implementados na classe **CandidatoEmprego.java** :

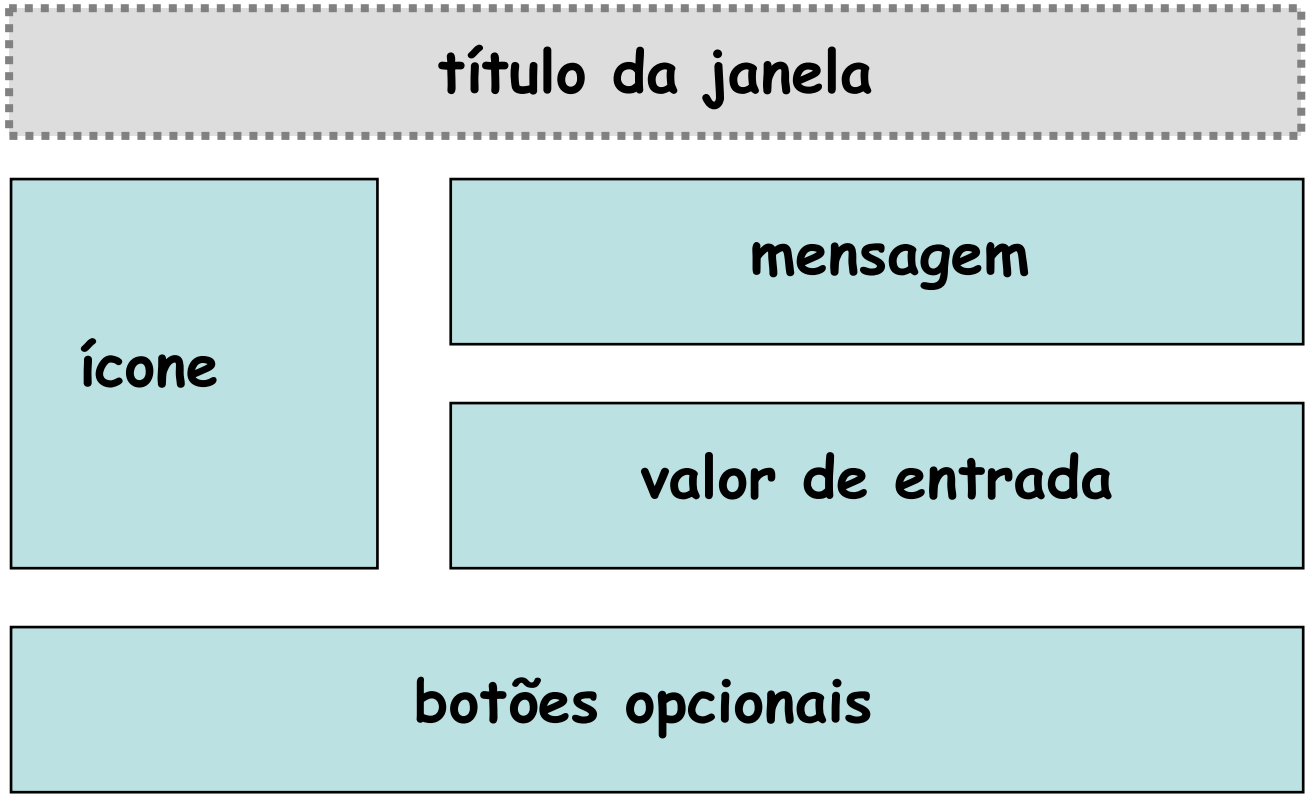


A classe JOptionPane

- Através da classe JOptionPane é fácil exibir uma caixa de diálogo padrão que induza o usuário a um valor ou o informa de alguma coisa.
- A classe aparenta ser complexa devido ao grande número de métodos, porém a quase totalidade das chamadas é constituída de métodos **estáticos**.
- Na sua forma mais básica, ela exibe uma linha de mensagem e um botão rotulado com “OK”.
- Uma vez que o diálogo tenha sido exibido, ele fica na tela até o usuário pressionar o botão “OK”. Enquanto o diálogo continuar na

A classe JOptionPane

- A aparência de uma das caixas de diálogo é similar à figura abaixo:



A classe JOptionPane

Constantes para tipos de mensagens:

ERROR_MESSAGE
INFORMATION_MESSAGE
WARNING_MESSAGE
QUESTION_MESSAGE
PLAIN_MESSAGE

Constantes para opções de botões:

YES_NO_OPTION

YES_NO_CANCEL_OPTION
OK_CANCEL_OPTION

Constantes inteiras de retorno:

JOptionPane.YES_OPTION
JOptionPane.NO_OPTION
JOptionPane.CANCEL_OPTION
JOptionPane.OK_OPTION

JOptionPane - exemplos 1

Mostra um diálogo de erro com título “Mensagem de erro” e mensagem “Entrada inválida”:

```
JOptionPane.showMessageDialog(null,
  “Entrada inválida”, “Mensagem de erro”,
  JOptionPane.ERROR_MESSAGE);
```

Mostra um diálogo com as opções yes/no e a mensagem “Escolha uma opção”:

```
JOptionPane.showConfirmDialog(null,
  “Escolha uma opção", “Diálogo de escolha",
  JOptionPane.YES_NO_OPTION);
```

JOptionPane - exemplo 2

Mostra um diálogo de aviso com opções OK e CANCELA:

```
Object[] options = { "OK", "CANCELA" };
JOptionPane.showOptionDialog(null, “Continua?",
   "Warning", JOptionPane.YES_NO_OPTION,
   JOptionPane.WARNING_MESSAGE, null, options,
   options[0]);
```

Assinatura do método:

```
showOptionDialog(Component parent, Object mensagem,
                 String título, int optionType,
                 int messageType, Icon icone,
                 Object[] options,
                 Object intialValue).
```

JOptionPane - exemplos 3

Mostra um diálogo que requer um String de entrada:

```
String inputValue = JOptionPane.showInputDialog(
                    “Por favor, entre o valor:");
```

```
Object[] possibleValues = { “Um", “Dois", “Três" };
Object selectedValue = JOptionPane.showInputDialog(
                    null,“Escolha um", "Input",
                JOptionPane.INFORMATION_MESSAGE, null,
                possibleValues, possibleValues[0]);
```

JFileChooser (tópico extra)

- Diálogos para selecionar arquivos.
- São herança direta de JComponent.
- Os principais tipos são : showDialog, showOpenDialog e showSaveDialog.
- Simples de serem criados:
  - JFileChooser chooser = new JFileChooser();
  - int result = chooser.showDialog(parentFrame);
- result :
  - JFileChooser.APPROVE_OPTION (0) se algum arquivo é escolhido.
    - JFileChooser.CANCEL_OPTION (1) se cancel é escolhido ou a janela de diálogo é fechada.

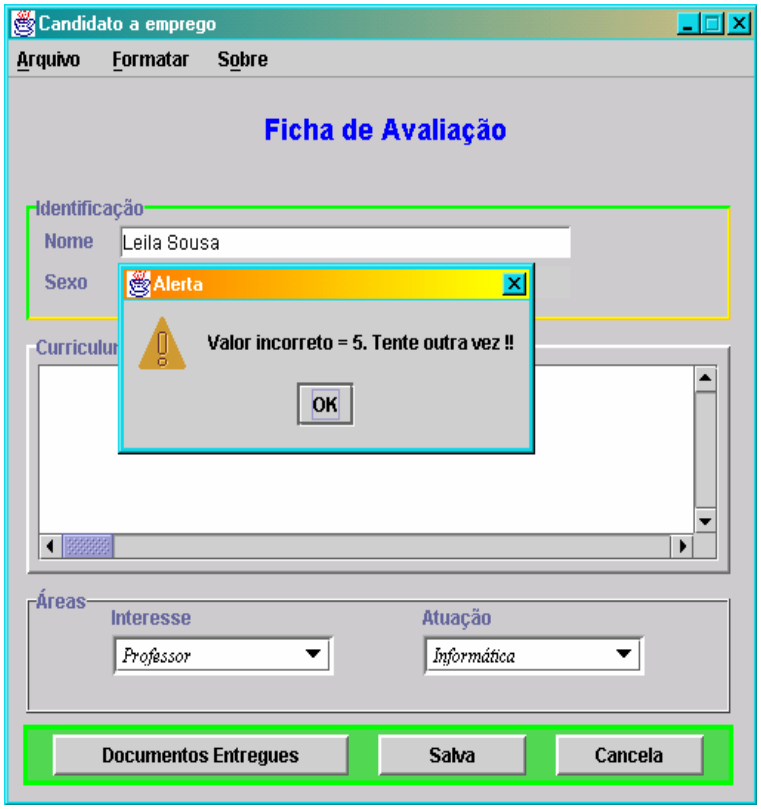
JFileChooser – execute e veja o resultado !!

```
// File Chooser: Exemplo Simples
import javax.swing.*;
import java.io.*;

public class FileChooserExample1 {
        public static void main(String[] args) {
                JFrame parentFrame = new JFrame("File Chooser Example 1");
                JFileChooser chooser = new JFileChooser();
                int result = chooser.showDialog(parentFrame, “Título”);
                File selectedFile = chooser.getSelectedFile();
                System.out.println("Return value from showDialog is " + result);
                if (selectedFile != null)
                {
                    System.out.println("Chosen file is " + selectedFile.getPath());
                }
                else
                {
                    System.out.println("No file was selected");
                }
                System.exit(0);         }         }
```

JOptionPane - exercício 1 *LFSC*

Componentes a serem implementados no método **main** da classe **CandidatoEmprego.java** :

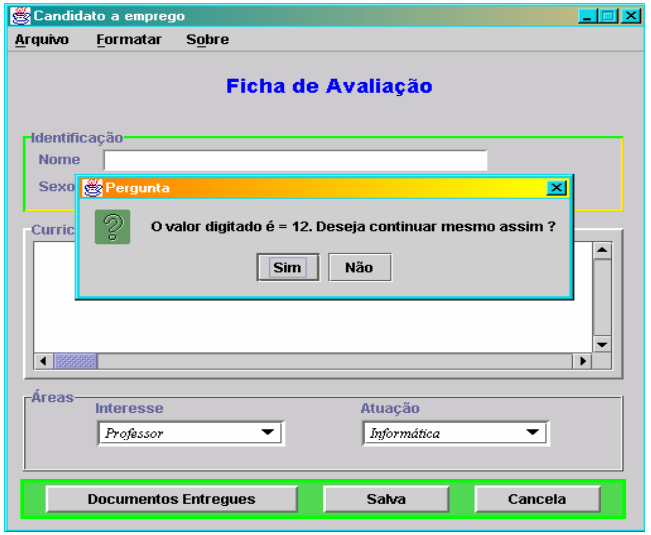
- Chamar o método showMessageDialog da classe JOptionPane com os seguintes parâmetros:
  - **Frame-pai**: nulo.
  - **Mensagem**: "Valor incorreto = " + valor + ". Tente outra vez !!".
  - **Título**: "Alerta".
  - **Tipo de mensagem**: JOptionPane.WARNING_MESSAGE.



* Passar o parâmetro pela linha de comando e mostrar a caixa de diálogo se esse valor for menor que 10.

JOptionPane - exercício 2 *LFSC*

Componentes a serem implementados no método **main** da classe **CandidatoEmprego.java** :



- Chamar o método showOptionDialog da classe JOptionPane com os seguintes parâmetros:
  - **Frame-pai**: nulo.
  - **Mensagem**: "O valor digitado é = " + valor ". Deseja continuar mesmo assim ?".
  - **Título**: "Pergunta".
  - **Tipo de opção**: JOptionPane.YES_NO_OPTION.
  - **Tipo de mensagem**: JOptionPane.QUESTION_MESSAGE.
  - **Ícone**: nulo.
  - **Array de opções**: array de Object com as opções "Sim" e "Não".
  - **Opção default**: "Sim" (posição zero).

* Passar o parâmetro pela linha de comando e mostrar a caixa de diálogo se esse valor for maior que 10.

JOptionPane - exercício 3 *LFSC*

Componentes a serem implementados no construtor da classe **CandidatoEmprego.java** :

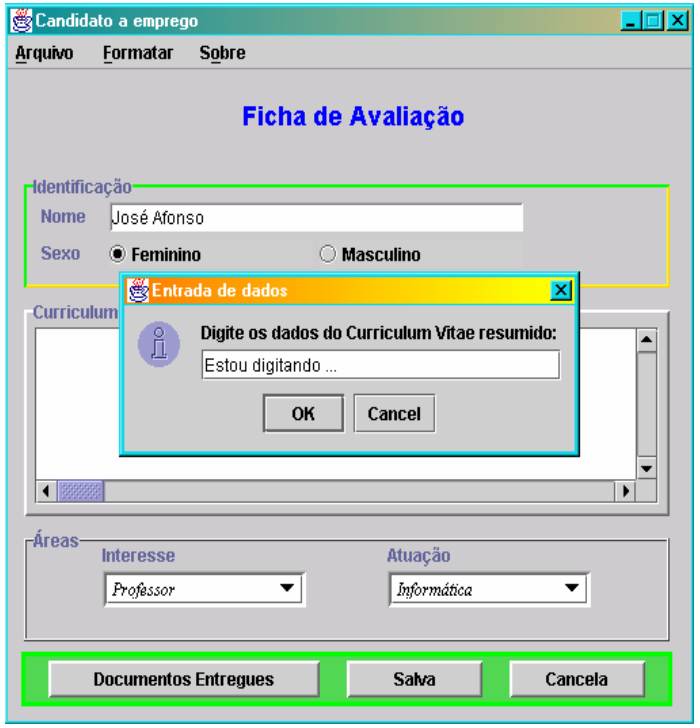
- Após mostrar a janela, chamar o método showInputDialog da classe JOptionPane com os seguintes parâmetros:
  - **Frame-pai**: janela.
  - **Mensagem**: "Escolha o nome do candidato ao emprego: ".
  - **Título**: "JOptionPane com ComboBox".
  - **Tipo de mensagem**: JOptionPane.PLAIN_MESSAGE.
  - **Ícone**: tips.gif.
  - **Array de opções**: array de Object com os nomes "Antônio da Siva, José Afonso e Ana Rita".
  - **Opção default**: "Ana Rita" (posição 2).

*Capturar o retorno como String e passá-lo ao método setNome(String s) para que o campo jtfNome seja atualizado com o valor escolhido (setText( )).

JOptionPane - exercício 4 *LFSC*

Componentes a serem implementados no construtor da classe **CandidatoEmprego.java** :

- Após o exercício anterior, invocar novamente o método showInputDialog da classe JOptionPane com os seguintes parâmetros:
  - **Frame-pai**: janela.
  - **Mensagem**: "Digite os dados do Curriculum Vitae resumido: ".
  - **Título**: "Entrada de dados".
  - **Tipo de mensagem**: JOptionPane.INFORMATION_MESSAGE.



*Capturar a String de retorno e passá-la ao método setCurriculum(String s) para que o campo jtaCurriculum seja atualizado com o valor escolhido (setText( )).

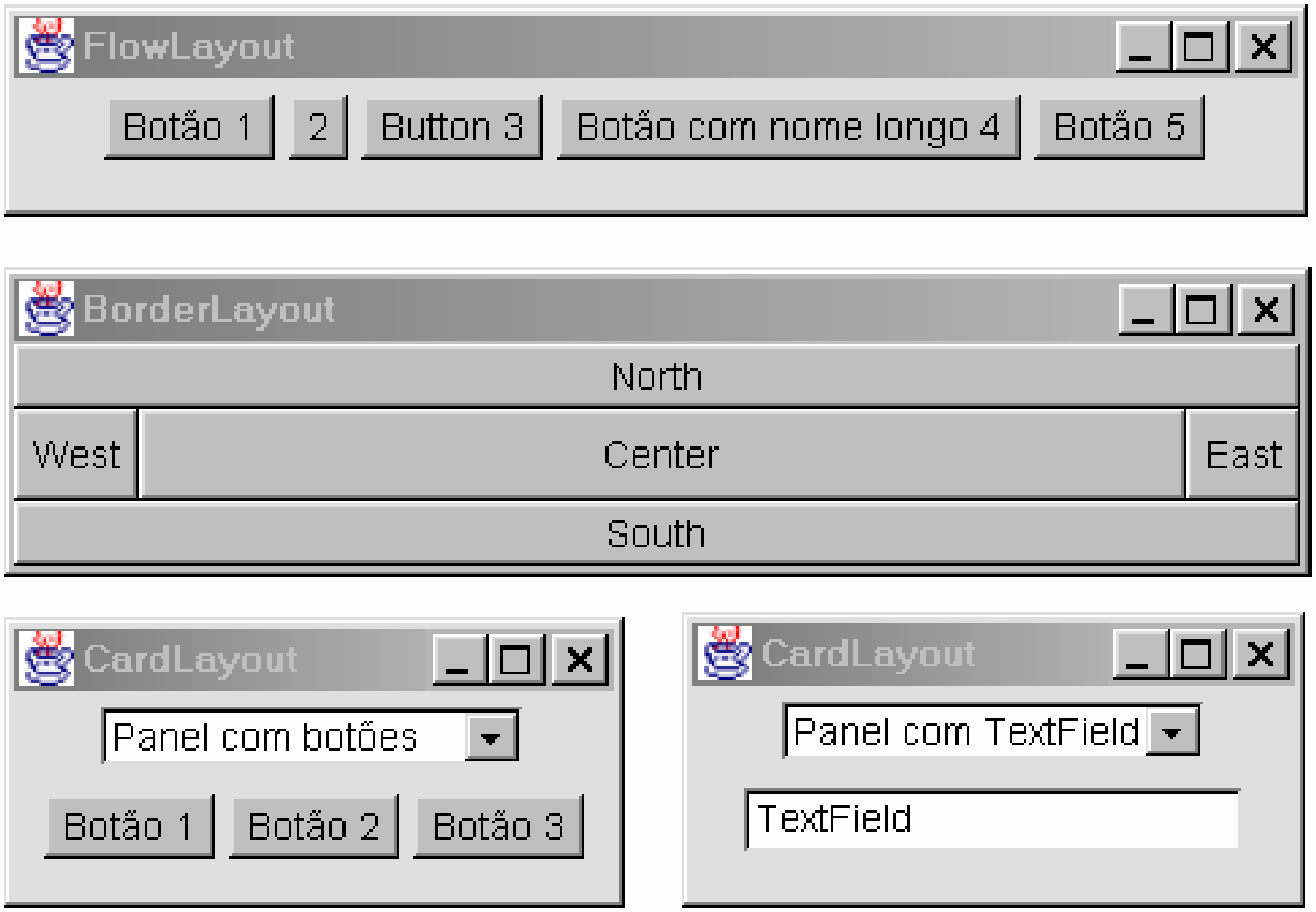
Gerenciadores de Layout

- Embora seja possível posicionar objetos em coordenadas fixas (x, y), deve-se evitar essa prática em Java.
- Isso causa problemas de portabilidade: outras JVMs, GUIs e *Window Managers* terão fontes e outros elementos em tamanhos e proporções diferentes.
- Baseados em *regras,* os *Layout Managers* calculam as coordenadas automaticamente.

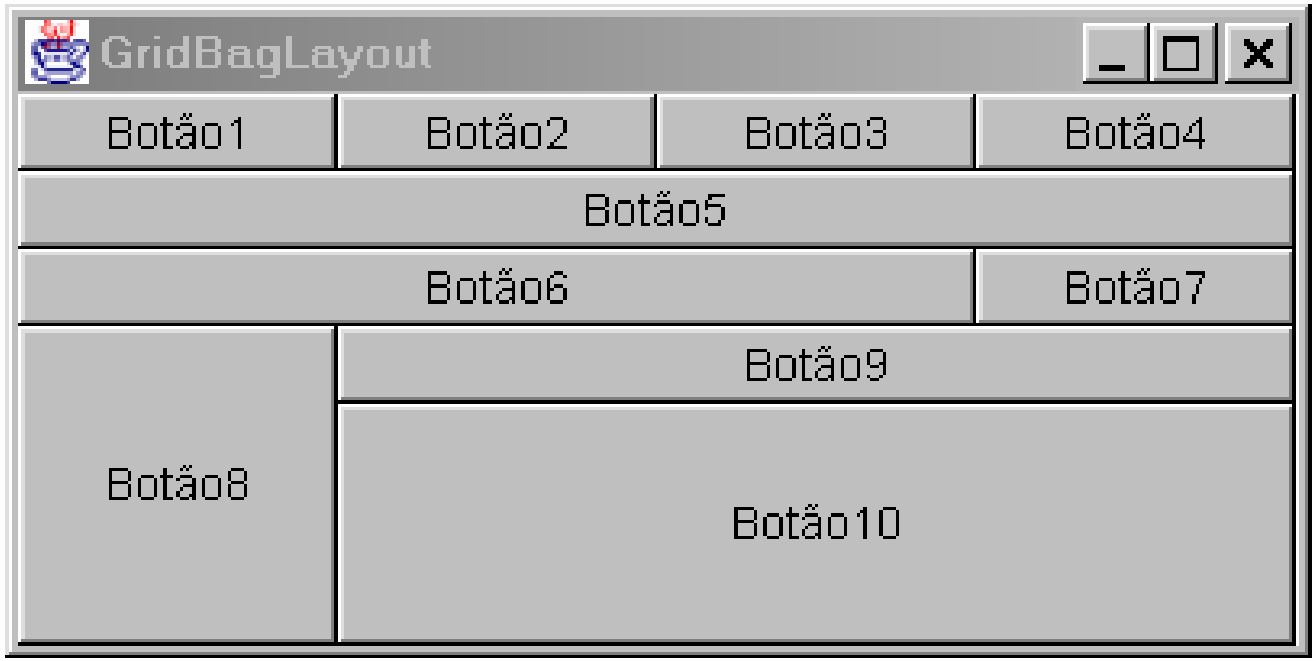
Gerenciadores de Layout

- Há 6 gerenciadores de layout pré-definidos no pacote `java.awt.*`:
  - **FlowLayout**
  - **GridLayout**
  - **BorderLayout**
  - CardLayout
  - GridBagLayout
  - BoxLayout
- Implicitamente, cada container possui um gerenciador de layout default associado. Assim, se não for declarado nenhum gerenciador de layout para o container, será usado o layout default.
- Um programador pode também criar novos gerenciadores de layout.

Gerenciadores de Layout



Gerenciadores de Layout



Escolhendo um gerenciador de layout

- Se você precisa mostrar alguns componentes utilizando os seus tamanhos naturais em uma linha, então use o `FlowLayout`.
- Se você precisa mostrar componentes de mesmo tamanho em linhas e colunas, então use o `GridLayout`.
- Se você precisa mostrar painéis alternativamente, então use o `CardLayout`.
- Se você precisa mostrar componentes e espalhá-los utilizando todo espaço disponível, então utilize o `BorderLayout` ou `GridBagLayout`.

Gerenciadores de layout - **regras gerais**

- Cada container tem um gerenciador de layout default:
  - JPanel (incluindo JApplet) → FlowLayout
  - Window → BorderLayout
  - JMenuBar → BoxLayout
- Para utilizar um gerenciador de layout diferente do default, você precisa criar uma instância do gerenciador de layout desejado e avisar ao container para usá-lo:

```
aContainer.setLayout(new CardLayout());
```

ou

```
CardLayout meuCardLayout = new CardLayout();
aContainer.setLayout(meuCardLayout);
```

ou

```
aContainer.setLayout(null); //sem gerenciador
```

Gerenciadores de layout - **regras gerais**

- Os métodos `add`, `remove` e `removeAll` (que adiciona e removem componentes) podem ser utilizados a qualquer momento.
- Se você muda o tamanho de um componente indiretamente (por exemplo, mudando a sua `Font`), então você deverá invocar o método `invalidate` para o componente e, logo, após invocar o método `validate` para o container.
- Os métodos `getPreferredSize` e `getMinimumSize` podem retornar valores sem significado (os seus valores são válidos somente após o componente ter sido mostrado a primeira vez).

Gerenciadores de layout - **curiosidades** *LFSC*

- Pode-se simplificar um layout pelo agrupamento de componentes em containers do tipo JPanel (visíveis ou invisíveis).

  ***Exemplo** : distribuir componentes no ContentPane de um JFrame.*

- Pode-se usar painéis dentro de painéis e cada um deles sendo controlado por um gerenciador de layout diferente, o que também auxilia a distribuição dos componentes visuais.

  ***Exemplo** : botões posicionados com distâncias horizontais diferentes.*

- É possível atribuir painéis às áreas dos gerenciadores de layout (os que assim o permitam) e aplicar a cada uma dessas áreas um novo gerenciador.

  ***Exemplo** : quando não se consegue um boa distribuição direta (sem JPanel) dos componentes na área disponibilizada por*

FlowLayout

- FlowLayout é o layout default em painéis e applets.
- À medida que os componentes são acrescentados ao layout, eles são posicionados em uma linha da esquerda para a direita.
- O tamanho para cada componente é determinado automaticamente pelo gerenciador de layout (através da noção de *preferred size*).
- Uma nova linha é automaticamente criada quando falta espaço na linha corrente.

FlowLayout

- Dentro de cada linha, os componentes são automaticamente centralizados (por default) ou alinhados à direita ou esquerda, quando especificado:

```
FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
setLayout(layout);
layout.setAlignment(FlowLayout.RIGHT);
validate();
```

- O programador pode especificar o tamanho do espaço entre os componentes, tanto vertical quanto horizontalmente:

```
FlowLayout layout = new FlowLayout(FlowLayout.CENTER, 5, 5));
layout.setHgap(10);
layout.setVgap(10);
validate();
```

FlowLayout - exemplo

```
import java.awt.*;
import javax.swing.*;
public class Janela extends JFrame
{
   public static void main(String args[])
   {
      Janela      j = new Janela();
      Container   c = j.getContentPane();
      c.setLayout (new FlowLayout());
      c.add (new Button ("Eu"));
      c.add (new Button ("penso,"));
      c.add (new Button ("logo"));
      c.add (new Button ("eu existo!"));
      j.pack();
      j.setVisible(true);
   }
}
```

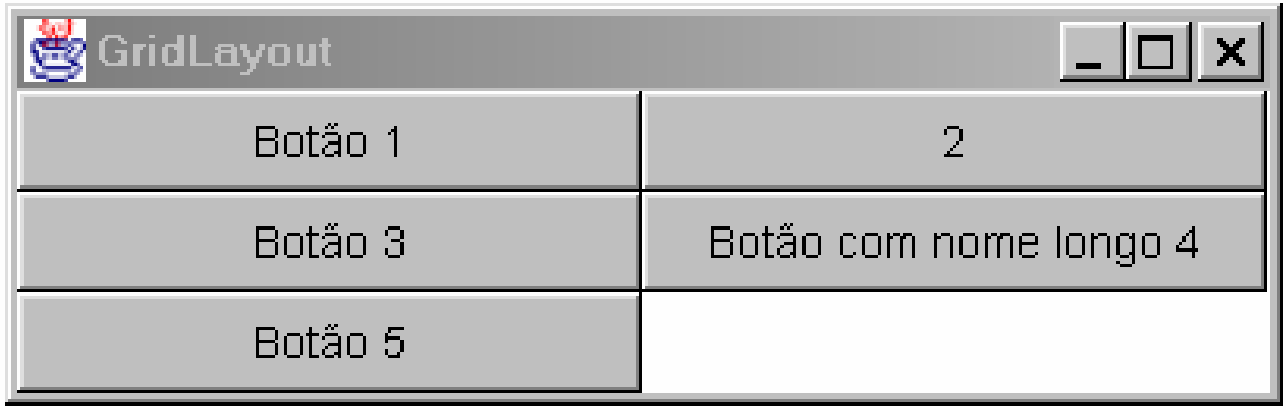
FlowLayout - exercício *LFSC*

- Considere a classe CandidatoEmprego.java. Para o seu **painelBotoes**:

  - Coloque entre comentários a linha com o método painel.setLayout(null).
  - Determine o novo layout : FlowLayout(...), com alinhamento centralizado, espaço horizontal 10 e espaço vertical 5.
  - Coloque entre comentários as linhas com o método setBounds(...) dos botões. **Não** retirar o setBounds( ) do próprio painel.
  - **Crie 3 novos botões** (Anterior, Próximo e Novo) e os adicione ao painel.

FlowLayout – resultado final *LFSC*

GridLayout



- Componentes são colocados em um matriz com um numero específico de linhas e colunas.
- Cada componente ocupa exatamente uma célula da matriz.
- As células são preenchidas da esquerda para a direita e do topo para a base.
- Todas as células da matriz são do mesmo tamanho.

GridLayout - exemplo

```
import java.awt.*;
import javax.swing.*;
public class Janela extends JFrame
{
   public static void main(String args[])
   {
      Janela      j = new Janela();
      Container   c = j.getContentPane();
      c.setLayout (new GridLayout(2,3));
      c.add (new Button ("Eu"));
      c.add (new Button ("penso,"));
      c.add (new Button ("logo"));
      c.add (new Button ("eu"));
      c.add (new Button ("existo!"));
      j.pack();
      j.setVisible(true);
   }
}
```

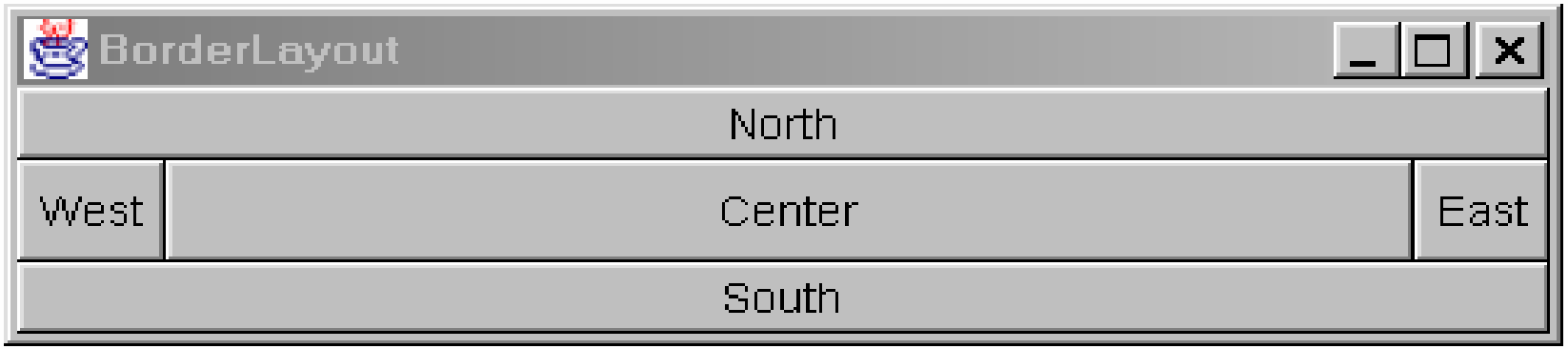
GridLayout - exercício 1 *LFSC*

- Considere a classe DocumentosEntregues.java:
  - Coloque entre comentários a linha com o método setLayout(null) do ContentPane da classe atual.
  - Determine o novo layout : GridLayout(...), com 5 linhas, 1 coluna, espaço horizontal 0 e espaço vertical 4.
  - Coloque entre comentários as linhas com o método setBounds(...) dos cinco JCheckBox´s.

GridLayout - exercício 2

- Crie um JFrame com 6 botões distribuídos por um gerenciador de layoute GridLayout. Os botões são inicialmente organizados em uma matriz com 2 linhas e 3 colunas. Quando o usuário clica em um dos botões, o layoute é mudado para um matriz de 3 linhas e 2 colunas. Ao clicar novamente, a matriz é transformada para 2 linhas e 3 colunas. E, assim, sucessivamente.

BorderLayout



- Cria 5 áreas pré-definidas, onde um componente ou um grupo de componentes pode ser colocado: NORTH, SOUTH, EAST, WEST e CENTER.
- O programador especifica a área na qual um componente deve aparecer.
- As dimensões relativas das áreas são governadas pelo tamanho dos componentes inseridos no layout.
- **Dica:** colocar o componente com maior tamanho no **centro**

BorderLayout - exemplo

```
import java.awt.*;
import javax.swing.*;
public class Janela extends JFrame
{
   public static void main(String args[])
   {
      Janela      j = new Janela();
      Container   c = j.getContentPane();
      c.setLayout (new BorderLayout());
      c.add (new Button ("Eu"),BorderLayout.NORTH);
      c.add (new Button ("penso,"),BorderLayout. SOUTH);
      c.add (new Button ("logo"),BorderLayout.EAST);
      c.add (new Button ("eu"),BorderLayout.WEST);
      c.add (new Button ("existo"),BorderLayout.CENTER);
      j.pack();
      j.setVisible(true);
   }
}
```

BorderLayout - exercício 1 *LFSC*

- Considere a classe CandidatoEmprego.java. Para o seu **painelCurriculum**:
  - Coloque entre comentários a linha com o método setLayout(null).
  - Determine o novo layout : BorderLayout( ).
  - Coloque entre comentários a linha com o método setBounds(...) do JScrollPane.
  - Posicione o JScrollPane no "Centro" da área disponível do painel.

BorderLayout - exercício 2

- Crie um JFrame com 5 botões gerenciados por um BorderLayout, onde os botões estejam separados por 5 pixels.
- Quando o usuário clica em um botão, o mesmo se torna invisível, e o botão invisível (se houver algum) se torna visível.
- Dica: o método `validate` deve ser chamado sempre que os botões se tornam visíveis ou invisíveis.

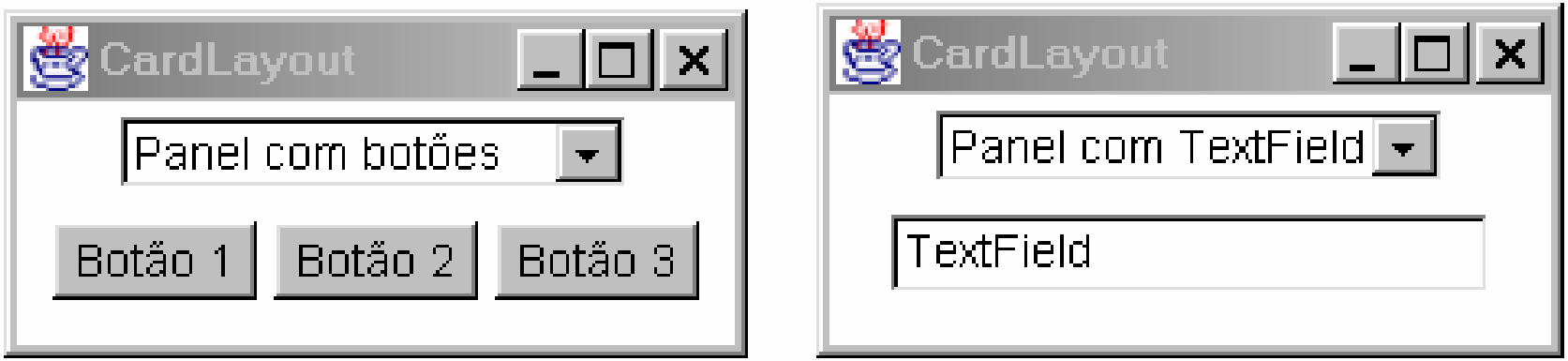
Gerenciadores de layout – exercício geral *LFSC*

- Considere a classe CandidatoEmprego.java.

Para todos os componentes, organize-os na tela apenas utilizando os gerenciadores de layout adequados:

- Coloque entre comentários todas as linhas com o método setLayout(null).
- Coloque entre comentários todas as linhas com o método setBounds(...).
- Determine os novos gerenciadores de layout, de modo que os componentes sejam devidamente dispostos na tela e apresentem a mesma organização anterior (quando se usava o setBounds( )).

CardLayout



- Componentes governados por um CardLayout são “empilhados” de tal forma que somente um componente é mostrado na tela.
- Componentes são ordenados de acordo com a ordem na qual eles são inseridos no container.
- Métodos controlam qual componente é visível no container.

CardLayout - exemplo

```
import java.awt.*;
import java.awt.event.*;
public class ExemploCardLayout extends Frame
                    implements ItemListener {
   Panel paineisAlternativos = new Panel();
   String botoes = "Panel com botões";
   String texto = "Panel com TextField";

   public ExemploCardLayout() {
      setLayout(new BorderLayout());
      Panel choicePanel = new Panel();

      Choice decisao = new Choice();
      decisao.addItem(botoes);
      decisao.addItem(texto);
      decisao.addItemListener(this);
      choicePanel.add(decisao);
      add("North", choicePanel);
```

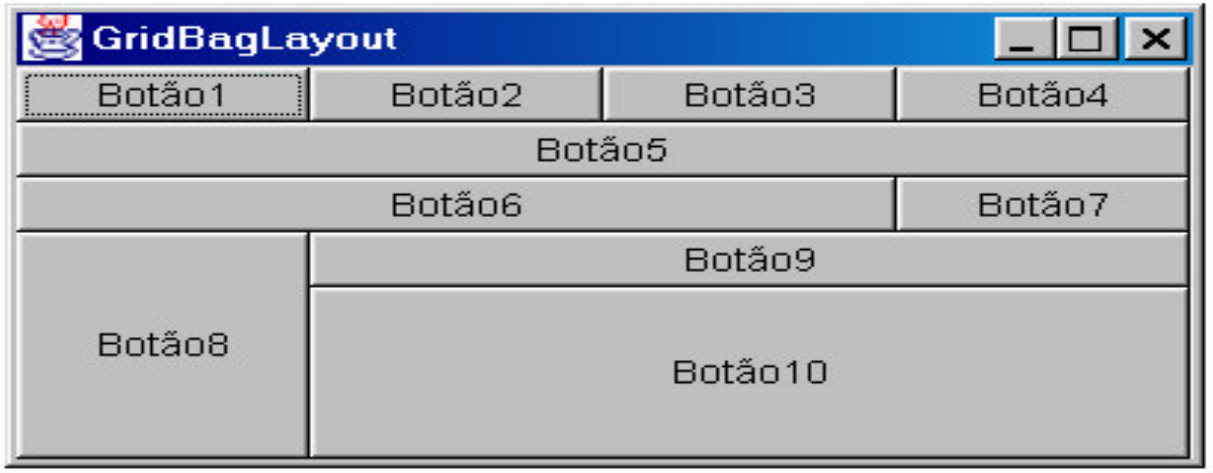
CardLayout - exemplo - cont.

```
    paineisAlternativos.setLayout(new CardLayout());

    Panel alt1 = new Panel();
    alt1.add(new Button("Botão 1"));
    alt1.add(new Button("Botão 2"));
    alt1.add(new Button("Botão 3"));

    Panel alt2 = new Panel();
    alt2.add(new TextField("TextField", 20));

    paineisAlternativos.add(botoes, alt1);
    paineisAlternativos.add(texto, alt2);
    add("Center", paineisAlternativos);
}
```

CardLayout - exemplo - cont.

```
 public void itemStateChanged(ItemEvent e)
 {
   CardLayout cl =
    (CardLayout)paineisAlternativos.getLayout();
   cl.show(paineisAlternativos, (String)e.getItem());
 }

 public static void main(String args[])
 {
   ExemploCardLayout window = new ExemploCardLayout();
   window.setTitle("ExemploCardLayout");
   window.pack();
   window.show();
 }
}
```

GridBagLayout



- O gerenciador de layout GridBagLayout é o mais versátil e mais complexo de todos os gerenciadores de layout predefinidos.
- Projetado com uma matriz bi-dimensional de colunas e linhas.
- Contudo, nem todas as células na matriz são do mesmo tamanho.
- Componentes podem se expandir em várias colunas e linhas.
- Cada componente em um GridBagLayout é associado com um conjunto de restrições definidas pela classe GridBagConstraints.

GridBagLayout

- Cada linha pode ter um tamanho próprio
- Cada coluna pode ter um tamanho próprio
- O número de linhas e de colunas é determinado dinamicamente
- GridBagLayout usa o preferred size dos componentes para determinar o tamanho de cada célula
- O tamanho e a posição de cada componente são determinados por um GridBagConstraints, através do método (da classe GridBagLayout) setConstraints.

GridBagLayout - exemplo

```
Panel p = new Panel();
GridBagLayout g = new GridBagLayout();
p.setLayout(g);
GridBagConstraints c = new GridBagConstraints();

.   .   .      (atribui as restrições)

Label labelRua =  new Label("Rua:", Label.LEFT);
g.setConstraints(labelRua,c);
p.add(labelRua);
```

GridBagConstraints

- **gridwidth e gridheight**: especifica o número de linhas e colunas na área de display. Default = 1.

```
GridBagConstraints c = new GridBagConstraints();
c.gridwidth = 2;
c.gridheight = 1;
```

- **anchor**: especifica o alinhamento: CENTER, NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST.

```
c.anchor = GridBagConstraints.WEST;
```

GridBagConstraints

- Usar GridBagConstraints.REMAINDER para especificar que o componente é o último da linha ou coluna.

```
c.gridwidth = GridBagConstraints.REMAINDER;
```

- **fill:**Usado para indicar que os componentes devem utilizar (ou não) todo espaço da área de display. Valores: NONE (default), HORIZONTAL, VERTICAL e BOTH.

```
c.fill = GridBagConstraints.BOTH;
```

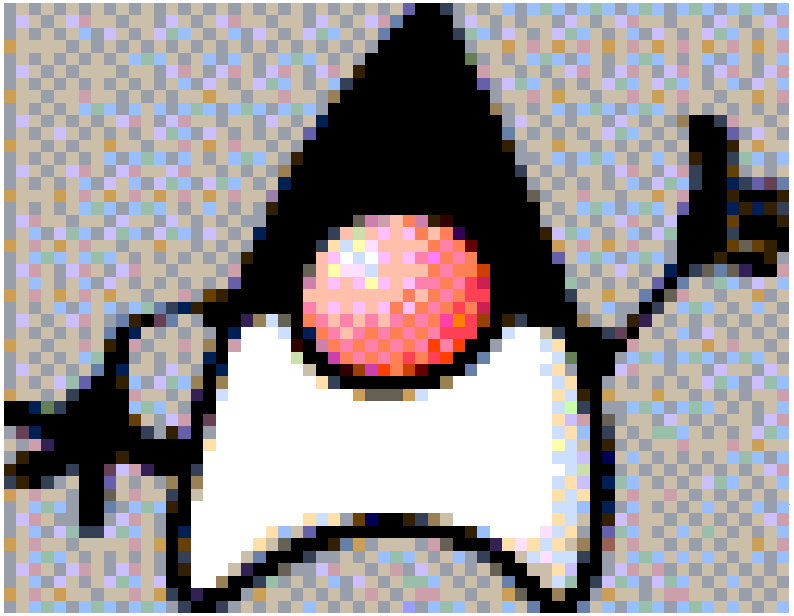
GridBagConstraints

- **insets:** indica a separação entre os componentes, através de um objeto Insets.

- Insets(int top, int left, int bottom, int right)

```
c.insets = new Insets(5, 5, 5, 5);
```

Eventos AWT



O que são eventos?

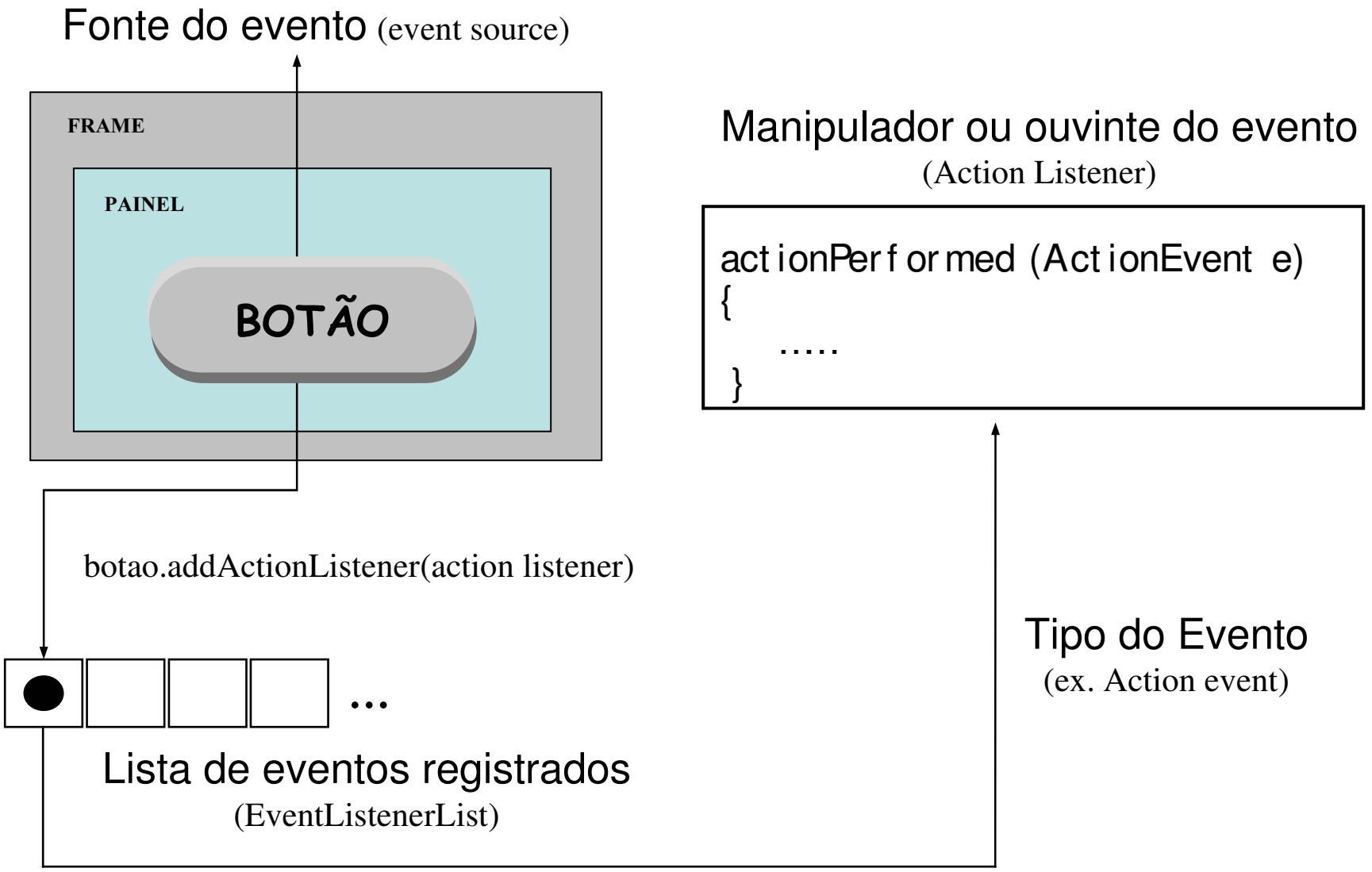
- Quando o usuário interage com uma interface de aplicativo final (teclado, mouse etc), isso causa o disparo de alguma ação: **o evento**.
- Mais propriamente, eventos são objetos que descrevem a ação ocorrida.
- Qualquer objeto pode ser notificado de um evento.
- Existem vários tipos de classes de eventos para tratar as diversas categorias de ações desencadeadas pelo usuário final.

*Incluir a instrução **import java.awt.event.*** .

Como os eventos são processados?

- Através de um modelo de delegação de eventos.
  - ⇨ Os componentes da AWT implementam rotinas de manipulação de eventos (listener) com o objetivo de receber eventos.
  - ⇨ O componente que gera o evento é chamado de **fonte do evento** (event source).
  - ⇨ O componente que trata o evento é chamado de **manipulador de evento** (event handler) ou **ouvinte de evento** (event listener).
  - ⇨ Normalmente a manipulação do evento é delegada para uma classe separada do componente fonte desse evento.

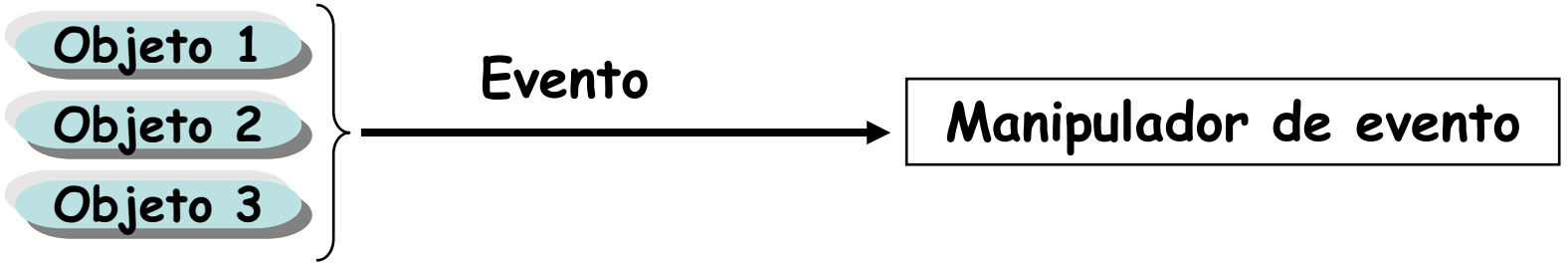
Esquema do modelo de delegação



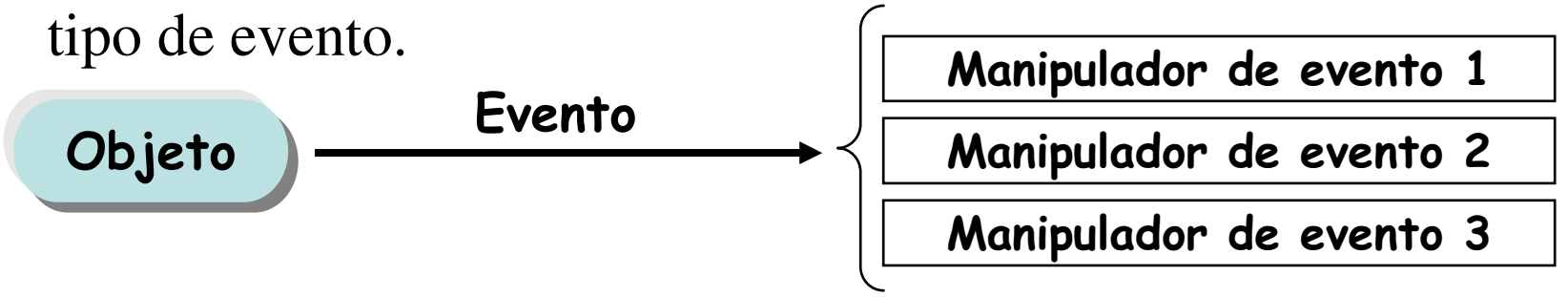
Modelo de delegação – possibilidades *LFSC*

1. Um objeto chama um único tipo de manipulador de evento.

2. Vários objetos chamam o mesmo manipulador de evento.



3. Um objeto chama vários manipuladores para tratar um único tipo de evento.



Modelo de delegação - exemplo

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

public class TesteEvento {
   public static void main (String args[])
   {
      JFrame j = new JFrame("Teste de evento");
      JButton b = new JButton("Pressione-me!");
      b.addActionListener(new ActionDemo());
      j.getContentPane().add(b, BorderLayout.CENTER);
      j.pack();
      j.setVisible(true);
   }
}

class ActionDemo implements ActionListener {
   public void actionPerformed (ActionEvent e){
      System.out.println("Acao executada");
   }
}//classe externa de tratamento do evento
```

Eventos gerados pelos componentes da AWT

- Cada componente da AWT possui um conjunto de tipos de eventos a ele associado.
- Os componentes que podem gerar eventos possuem métodos para adicionar ou remover "listeners".
  - ⇨addActionListener
  - ⇨removeActionListener
- A seguir são relacionados os tipos de eventos que cada componente da AWT pode gerar.

Alguns dos eventos gerados pelos componentes AWT/Swing

Componentes	Tipos de eventos que os componentes podem gerar										
	Action	Adjust.	Component	Container	Focus	item	key	mouse	mouse motion	text	window
button	✓		✓		✓		✓	✓	✓		
checkbox	✓		✓		✓	✓	✓	✓	✓		
combo box	✓		✓		✓	✓	✓	✓	✓		
Component			✓		✓		✓	✓	✓		
Container			✓	✓	✓		✓	✓	✓		
dialog			✓	✓	✓		✓	✓	✓		✓
frame			✓	✓	✓		✓	✓	✓		✓
label			✓		✓		✓	✓	✓		
list	✓		✓		✓	✓	✓	✓	✓		
menu item	✓										
panel			✓	✓	✓		✓	✓	✓		
radio button	✓		✓		✓	✓	✓	✓	✓		
scrollbar		✓	✓		✓		✓	✓	✓		
text area			✓		✓		✓	✓	✓	✓	
text field	✓		✓		✓		✓	✓	✓	✓	
window			✓	✓	✓		✓	✓	✓		✓

Mais detalhes em: http://java.sun.com/docs/books/tutorial/uiswing/events/eventsandcomponents.html.

Interface de eventos listener AWT

- Os tipos de eventos gerados pelos componentes da AWT são implementações da interface "listener".
- O nome da interface associado ao tipo de evento é definido como sendo o nome do evento, acrescido da palavra "Listener".

  Exemplo: tipo de evento: Action
  interface: ActionListener

- A seguir são descritas as interfaces dos eventos AWT.

Interface de eventos listener AWT (1)

Interface Listener	**Classe Adapter**	**Métodos**
ActionListener	----------	actionPerformed(ActionEvent)
AdjustmentListener	---------	adjustmentValueChanged(adjustmentEvent)
ComponentListener	ComponentAdapter	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentRezised(ComponentEvent) componentShown(ComponentEvent)
ContainerListener	ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener	FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
ItemListener	--------	itemStateChanged(ItemEvent)
KeyListener	KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener	MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseRealesed(MouseEvent)

Interface de eventos listener AWT (*2)

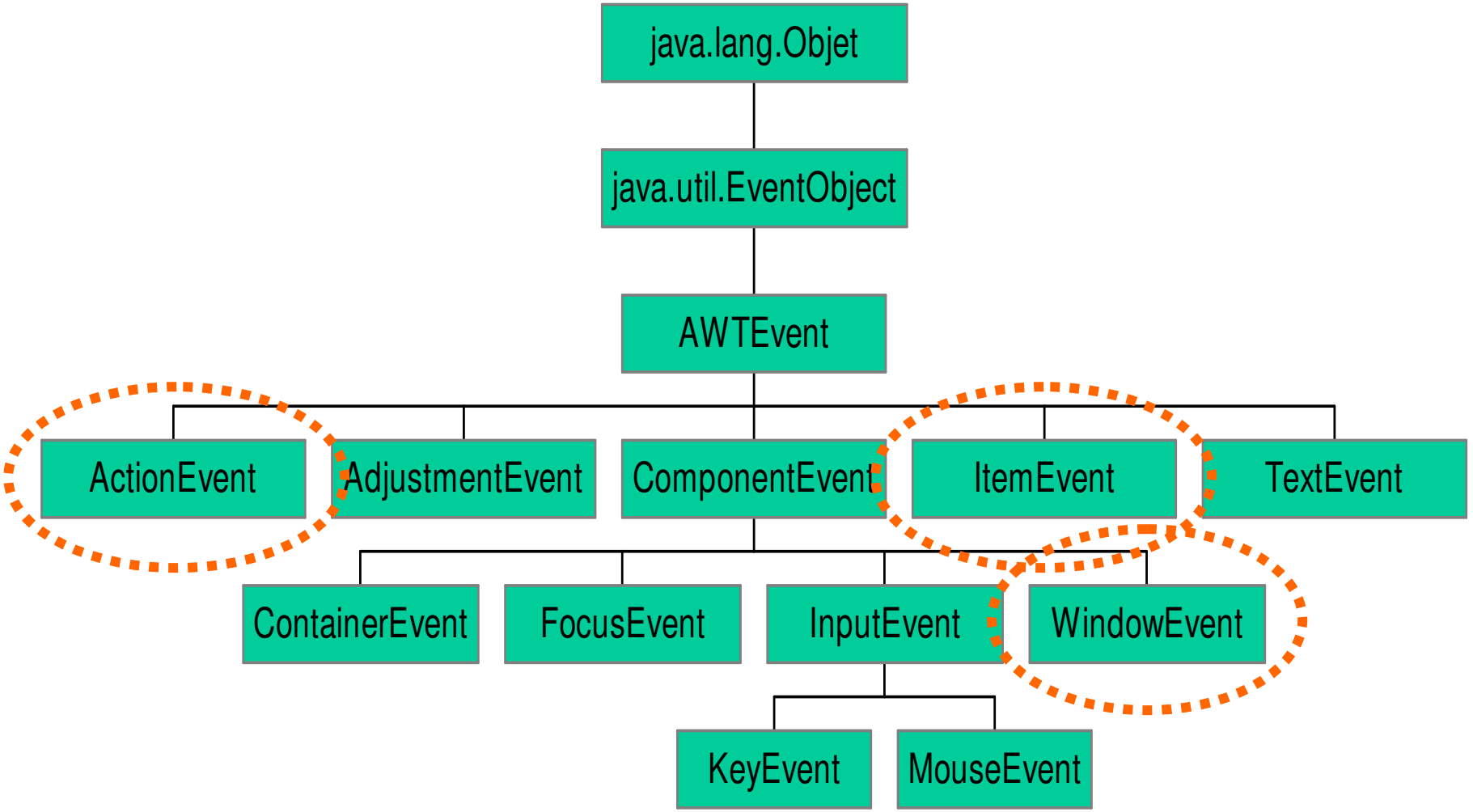
Interface Listener	**Classe Adapter**	**Métodos**
MouseMotionListener	MouseMotionAdapter	mouseDragged (MouseEvent) mouseMoved(MouseEvent)
TextListener	------	textValueChange(TextEvent)
WindowListener	WindowAdapter	windowActived(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

Mais detalhes em: http://java.sun.com/docs/books/tutorial/uiswing/events/api.html.

A hierarquia de classes de eventos AWT

- A classe AWTEvent
  - ⇨Cada método de cada interface “listener” dos eventos na AWT possui um único argumento: uma instância de uma classe que descende da classe java.awt.AWTEvent.
  - ⇨Essa classe não define nenhum método ou API que o usuário, usualmente, necessite. Contudo, herda um método muito útil da classe java.util.EventObject:
    - ***Object getSource( )*:** retorna o objeto (do tipo Object) que gerou o evento.
    - Sempre que possível, as subclasses da AWTEvent definem métodos similares que retornam tipos mais restritivos. Por exemplo, na classe ComponentEvent existe o método ***getComponent( )*** que retorna o componente que gerou o evento.

A hierarquia de classes de eventos AWT



As 4 maneiras de implementar um evento (1) *LFSC*

1. Fazer com que a própria classe (a que contém o objeto que gera o evento) implemente a interface (EventoListener) ou estenda a classe abstrata (EventoAdapter) do evento.

   Exemplo: **public class MinhaClasse implements WindowListener { }**

   ou

   public class MinhaClasse extends WindowAdapter { }.

2. Construir uma classe externa que implemente a interface ou estenda a classe abstrata do evento.

   Exemplo: **public class MinhaClasse { … }**

   public class EventoExt implements WindowListener { }

   ou

   public class EventoExt extends WindowAdapter { }.

As 4 maneiras de implementar um evento (*2) *LFSC*

3. Construir uma classe membro interna que implemente a interface ou estenda a classe abstrata do evento.

   Exemplo:
```
public class MinhaClasse { …
        class EventoInt   implements  WindowListener { }
                ou
        class EventoInt   extends  WindowAdapter { }  }.
```

4. Construir uma classe anônima interna que implemente a interface ou a classe abstrata do evento.

   Exemplo:
```
public class MinhaClasse { …
    janela.addWindowListener( new WindowAdapter( )
    {
        public void windowClosing(WindowEvent   e)
        {
                System.exit(0);
        } });      }.
```

O que um evento requer *LFSC*

- **Primeiramente, deve-se implementar o evento, escolhendo uma das 4 opções mostradas nos dois slides anteriores**.
- **Adicionar uma instância do ouvinte de evento ao(s) componente(s) desejado(s)**, conforme a implementação escolhida (slides anteriores):
  1. *componente.addWindowListener(this);*
  2. *componente.addWindowListener(new EventoExt( ));*
  3. *componente.addWindowListener(new EventoInt( ));*
  4. *Já é adição e também implementação do evento.*
- **Implementar corretamente o método da interface (ou classe abstrata) que sabe tratar o evento**. *Deve-se optar por uma implementação leve para evitar problemas de performance, pois o Java usa a mesma event-dispatching thread para eventos e painting.*

Como escrever WindowListener

- São gerados por um objeto janela (Window) após a janela ser: aberta, fechada, iconizada, desiconizada, ativada e desativada.
- A interface WindowListener e a classe abstrata WindowAdapter possuem os métodos a seguir:
  - void windowOpened(WindowEvent)
    - é executado assim que a janela é mostrada pela primeira vez.
  - void windowClosing(WindowEvent)
    - é executado assim que o usuário solicita o fechamento da janela. Para fechar a janela, usa-se o método dispose ou setVisible(false).

Como escrever WindowListener

- void windowClosed(WindowEvent)
  - é chamada pela AWT após a janela tiver sido fechada.
- void windowIconified(WindowEvent)
- void windowDeiconified(WindowEvent)
  - é chamado pela AWT assim que a janela tenha sido iconizada ou deiconizada.
- Void windowActivated(WindowEvent)
- void windowDeactivated(WindowEvent)
  - é chamado pela AWT assim que a janela tenha sido ativada ou desativada.

WindowListener – exemplo

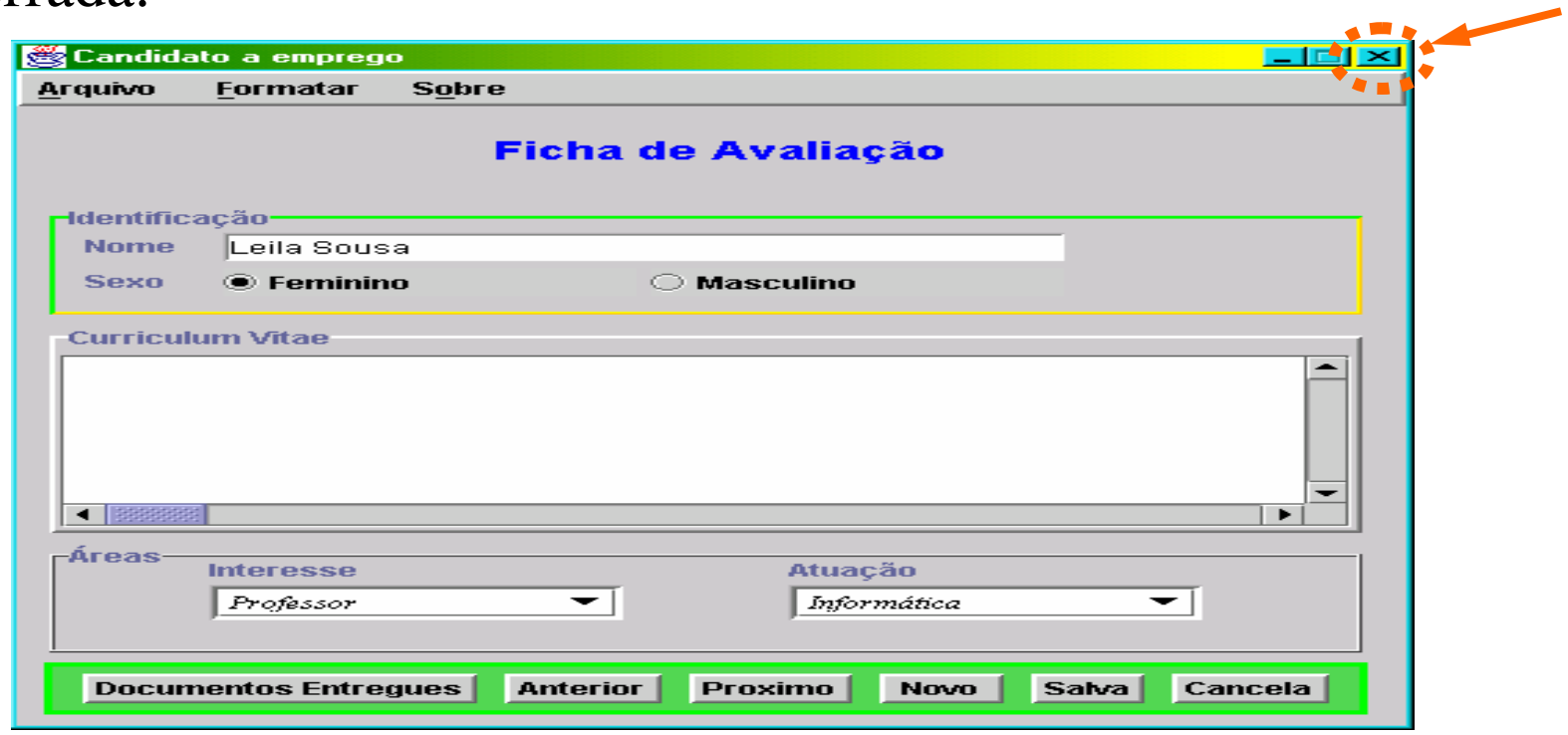
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class WindowListenerDemo {
   public static void main (String args[])
   {
      JFrame j = new JFrame("Evento de Janela");
      j.setSize(200,200);
      j.addWindowListener (new WindowAdapter() {
         public void windowClosing(WindowEvent e)
         {
            System.exit(0);
         }
      }); //classe anônima interna de tratamento do evento

      j.setVisible(true);
   }
}
```

A classe WindowEvent

- Cada método da WindowListener tem um único parâmetro: um objeto do tipo WindowEvent. A classe WindowEvent possui um método muito útil:
  - Window getWindow()
    - retorna a janela que gerou o evento.

WindowListener - exercício *LFSC*

Na classe CandidatoEmprego.java, implementar uma classe anônima interna de WindowAdapter que permita: **a)**fechar a janela atual (System.exit(0)) e **b)**emitir um alerta de que a aplicação será encerrada.



*Consulte o método **setDefaultCloseOperation(int i)** de JFrame para saber como proceder se o usuário retroceder na sua decisão de fechar a janela da aplicação.

Como escrever um **ActionListener**

- São os eventos mais fáceis e os mais comumente implementados.
- Implementa-se um ActionListener para responder a uma intervenção do usuário.
- Quando o usuário clicar um botão, o duplo clique em uma lista, a escolha em um menu ou pressionar o retorno de um campo texto, o gerenciador de eventos indicará que uma ação ocorreu.
  - ⇨ O resultado é o envio de uma mensagem "actionPerformed" a todos os ActionListeners que estão registrados no componente-fonte.

ActionListener para botões – exemplo (1)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ActionListenerDemo extends JFrame
   implements ActionListener {
    JButton b1, b2, b3;
    JPanel p1;
    static final String DISABLE = "disable";
    static final String ENABLE  = "enable";
    public ActionListenerDemo() {
        p1 = new JPanel();
        b1 = new JButton("Desativa o botão do meio");
        b1.setActionCommand(DISABLE);
        b2 = new JButton("Botão do meio");
        b3 = new JButton("Ativa botão do meio");
        b3.setEnabled(true);
        b3.setActionCommand(ENABLE);
        b1.addActionListener(this);
        b3.addActionListener(this);
```

ActionListener para botões – exemplo (*2)

```
        p1.add(b1);
        p1.add(b2);
        p1.add(b3);
        getContentPane().add(p1);
    }
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if (command == DISABLE)
                b2.setEnabled(false);
        else
                b2.setEnabled(true);
    }
    public static void main(String args[]) {
        ActionListenerDemo j = new ActionListenerDemo();
        j.setTitle("Vários Botões");
        j.pack();
        j.setVisible(true);
    }
} //Fim
```

A classe ActionEvent

- O método actionPerformed tem um parâmetro único: o objeto ActionEvent. A classe ActionEvent define dois métodos bastante úteis:

1. **String getActionCommand( )**
   - retorna o string associado com a ação. A maioria dos objetos que podem gerar ações suportam o método chamado setActionCommand que permite atribuir este string. Se o comando de ação não for atribuido explicitamente, então ele será o texto associado ao componente. Para objetos com múltiplos itens e, conseqüentemente, múltiplas ações possíveis, o comando ação é geralmente o nome do item selecionado.

A classe ActionEvent

2. **int getModifiers( )**
   - retorna um inteiro associado com a chave que foi pressionada pelo usuário quando o evento ocorreu. Pode se usar as constantes definidas na classe ActionEvent: SHIFT_MASK, CRTL_MASK, META_MASK e ALT_MASK para determinar qual chave foi pressionada.

ActionListener para texto

- Quando o usuário digita os dados em um JTextField ou JPasswordField e pressiona a tecla ENTER, um evento de ação ocorre.

- JTextArea não possui eventos de ação como JTextField. Algum evento externo é que costuma determinar quanto o texto nessa área deve ser processado.

ActionListener para texto – exemplo (1)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ActionListenerDemo extends JFrame {

    JTextField textField;
    JTextArea  textArea;
    String newline;
    public ActionListenerDemo() {
       textField = new JTextField(20);
       textArea  = new JTextArea(5,20);
       textArea.setBackground(Color.blue);
       textArea.setEditable(false);
       Container c = getContentPane();
       c.add(textField, BorderLayout.NORTH);
       c.add(textArea, BorderLayout.CENTER);
       textField.addActionListener(new MeuEvento());
       newline = System.getProperty("line.separator");
    }
```

ActionListener para texto – exemplo (*2)

```
  class MeuEvento implements ActionListener {
      public void actionPerformed (ActionEvent evt) {
             String text = textField.getText();
             textArea.append (text + newline);
             textField.selectAll();
      }
  } //classe-membro interna de tratamento do evento

  public static void main(String args[])
   {
       ActionListenerDemo  j = new ActionListenerDemo();
       j.setTitle("ActionListener com Texto");
       j.pack();
       j.setVisible(true);
    }
} //Fim
```

ActionListener – exercício 1 *LFSC*

Na classe CandidatoEmprego.java, implementar uma classe-membro interna de ActionListener que permita chamar DocumentosEntregues.java por meio do botão jbDocumentosEntregues :

ActionListener – exercício 2 *LFSC*

Quanto aos demais botões:

- **Anterior, Próximo, Novo e Salva** : implementar uma classe externa de ActionListener que emita a mensagem "Botão + TextoDoBotão + ainda não implementado", título "Informação" e tipo Informação, por meio de JOptionPane. showMessageDialog( ). Após isso, atribuir uma instância da classe de evento a cada um dos botões relacionados.
- **Salva** : implementar uma classe externa de ActionListener que emita a mensagem "Você clicou no ClasseDoBotão + TextoDoBotão", título "Informação" e tipo Informação, por meio de showMessageDialog( ). Após isso, atribuir também uma instância desta classe ao botão acima.
- **Cancela** : implementar uma classe anônima interna de ActionListener que permita fechar a janela (System.exit(0)). Após isso, atribuir uma instância desta classe ao botão acima.

* **Métodos a serem utilizados:** - evento.getActionCommand( ) e
- (evento.getSource( ).getClass( ).getName( )).

Como escrever um **ItemListener**

- Eventos em itens são gerados por componentes que implementam uma interface que tem a característica de permitir selecionar itens.
- Estes componentes mantém o estado, geralmente, ligado (on) ou desligado(off) para um ou mais itens. Os componentes na AWT que geram este tipo de evento são: checkboxes, checkbox menu itens, choices e lists.
- A interface ItemListener contém um único método.
  - ⇨ void itemStateChanged(ItemEvent)
    - é chamado pela AWT após a ocorrência de uma mudança de estado no componente.

A classe ItemEvent

- O método itemStateChanged possui um único parâmetro: um objeto ItemEvent . A classe ItemEvent possui os seguintes métodos:
  - **Object getItem( )**
    - retorna o componente associado ao item cujo estado tenha mudado. Freqüentemente, o retorno é um String contendo um texto sobre o item selecionado. No caso de uma lista de itens (List), ele é o inteiro (Integer) que especifica o índice do item selecionado.
  - **ItemSelectable getItemSelectable( )**
    - retorna o componente que gerou o evento (item).
  - **int getStateChange( )**
    - retorna o novo estado do item, que pode ser SELECTED e DESELECTED.

ItemListener – exemplo 1 (1)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ItemListenerDemo implements ItemListener {
  JComboBox lista;
  JLabel label;
  JFrame fr;
  public ItemListenerDemo() {
    fr = new JFrame("teste de ItemListener");
    Container c = fr.getContentPane();
    lista = new JComboBox();
    lista.addItem("primeiro");
    lista.addItem("segundo");
    lista.addItem("terceiro");
    lista.addItem("quarto");
    lista.addItemListener(this);
    label = new JLabel();
    setLabelText(lista.getSelectedIndex(),
                 lista.getSelectedItem().toString());
```

ItemListener – exemplo 1 (*2)

```
    c.add(lista,BorderLayout.CENTER);
    c.add(label,BorderLayout.SOUTH);
    fr.pack();
    fr.setVisible(true);  }
  void setLabelText(int num, String text) {
    label.setText("Item #" + num + " selecionado. "
                  + "Texto = \"" + text + "\".");  }
  public void itemStateChanged(ItemEvent e) {
    setLabelText(lista.getSelectedIndex(),
                  lista.getSelectedItem().toString());  }
  public static void main(String args[]) {
    ItemListenerDemo demo = new ItemListenerDemo(); }
}
```

ItemListener – exemplo 2 (1)

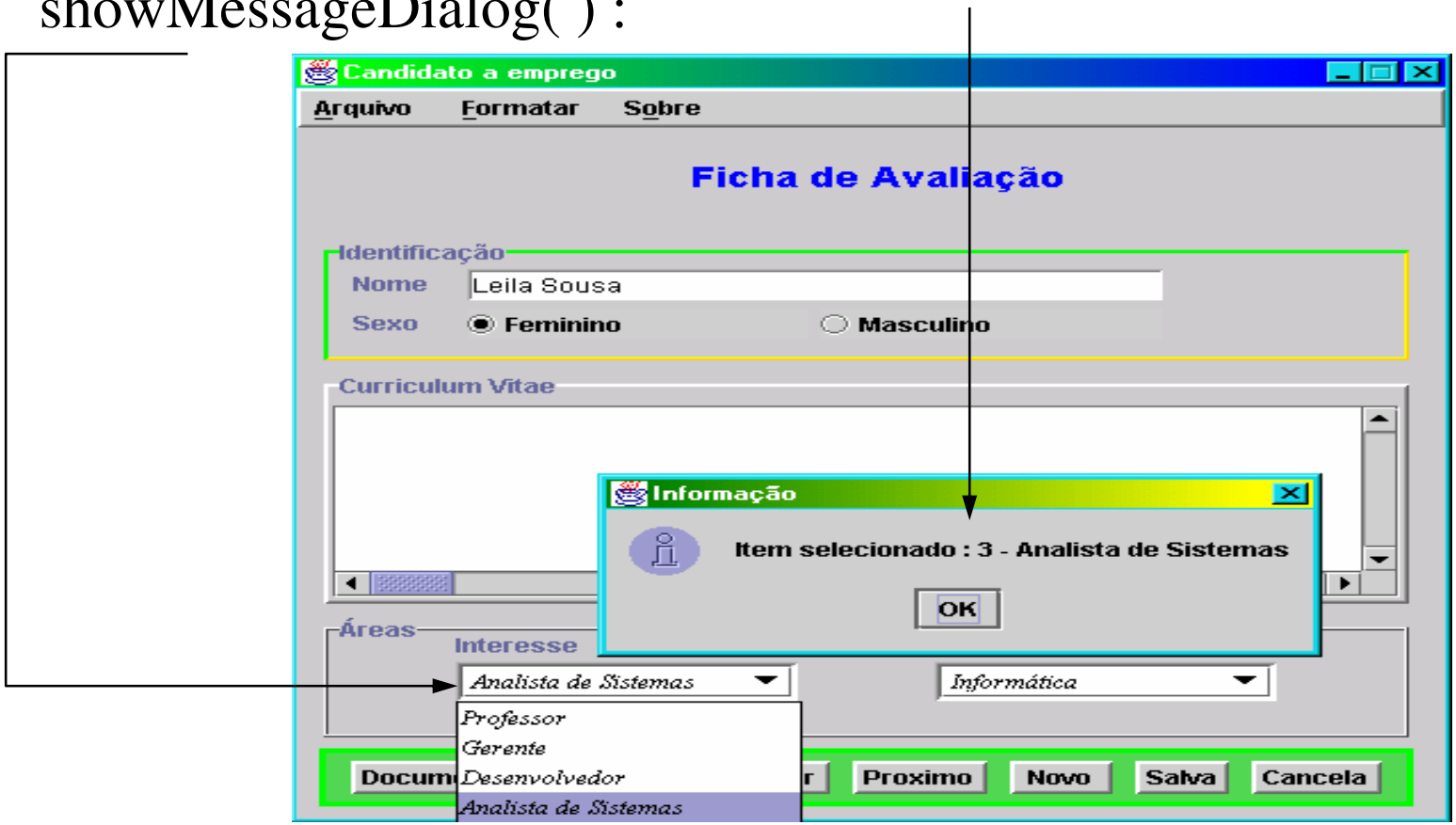
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ItemListenerDemo extends JFrame implements ItemListener {
  JPanel cards;
  final static String BUTTONPANEL = "Painel com botão";
  final static String TEXTPANEL = "Painel com texto";
  public ItemListenerDemo()
  {
    getContentPane().setLayout(new BorderLayout());
    setFont(new Font ("SansSerif", Font.PLAIN, 14));
    JPanel cp = new JPanel();
    Choice c = new Choice();
    c.add (BUTTONPANEL);
    c.add(TEXTPANEL);
    c.addItemListener(this);
    cp.add(c);
    getContentPane().add(cp, BorderLayout.NORTH);
    cards = new JPanel();
    cards.setLayout(new CardLayout());
    JPanel p1 = new JPanel();
```

ItemListener – exemplo 2 (*2)

```
    p1.add(new JButton("Botao 1"));
    p1.add(new JButton("Botao 2"));
    p1.add(new JButton("Botao 3"));
    JPanel p2 = new JPanel ();
    p2.add (new JTextField("TextField", 20));
    cards.add (BUTTONPANEL,p1);
    cards.add (TEXTPANEL,p2);
    getContentPane().add(cards,  BorderLayout.CENTER);
  }
  public void itemStateChanged(ItemEvent e) {
    CardLayout cl = (CardLayout) (cards.getLayout());
    cl.show (cards, (String) e.getItem());
  }
  public static void main(String args[]) {
    ItemListenerDemo j = new ItemListenerDemo();

    j.setTitle("ItemListener com CardLayout");
    j.pack();
    j.setVisible(true);
  }
}
```

ItemListener – exercício *LFSC*

Na classe CandidatoEmprego.java, implementar uma classe externa de ItemListener que permita mostrar o índice e o texto do item selecionado de jcbInteresse, por meio de JOptionPane. showMessageDialog( ) :



Como escrever um AdjustmentListener

- Os eventos “adjustment” notificam o usuário das mudanças nos valores dos componentes que implementam a interface “adjustable”. Os objetos “adjustable” possuem um valor inteiro. Eles geram eventos “adjustment” sempre que os valores mudarem. Na classe AWT, o único componente que implementa eventos “adjustable” é o Scrollbar.

Os 5 tipos de atributos da AdjustmentEvent

- Track
  - O usuário explicitamente tenha ajustado o valor do componente. Para um scrollbar, este valor deve ser o resultado do arrasto realizado pelo usuário na barra do scrollbar.
- Unit increment, unit decrement
  - o usuário fez um rapidíssimo (ou pequeníssimo) ajuste no valor do componente. Para um scrollbar, representa o resultado de um click em posição abaixo ou acima de um valor definido no scrollbar.

Os 5 tipos de atributos da AdjustmentEvent - cont.

- Block increment, block decrement
  - O usuário manifesta o desejo de ajustar o valor de um componente por uma grande quantidade. Para um scrollbar, é o resultado do clique acima ou abaixo do local onde está marcado, sendo uma grande porção.

O método da classe AdjustmentListener

- A interface AdjustmentListener contém somente um método, sem classe adapter.
  - Void adjustmentValueChanged(AdjustmentEvent)
    - é chamado após mudanças nos valores de componente que tratam “listened”.
    - A classe AdjustmentEvent possui os seguintes métodos que podem ser facilmente manuseados:
      - Adjustable getAdjustable ()
        - » retorna o componente que gerou o evento. Pode ser utilizado este no lugar do método getSource.
      - Int getAdjustmentType()
        - » retorna o tipo de “adjustment” que ocorreu.
      - Int getValue()

Exemplo: AdjustmentListener

```
import java.awt.event.*;
import java.awt.*;
public class Cores implements AdjustmentListener
{
   Scrollbar barraVermelha;
   Scrollbar barraVerde;
   Scrollbar barraAzul;
   Canvas canvas;
   Label infoLabel;
   public static void main(String[] args)
   {
      new Cores();
   }
   public Cores()
   {
      Frame f = new Frame("Selecione as cores");
      f.add("North", infoLabel = new Label("Cores"));
```

Exemplo: AdjustmentListener - cont.

```
f.add("Center", canvas = new Canvas());
canvas.setBackground(new Color(0,0,0));
Panel scrollBarPanel = new Panel();
scrollBarPanel.setLayout(new GridLayout(3,15));
scrollBarPanel.add(barraVermelha = new
      Scrollbar(Scrollbar.HORIZONTAL,0,0,0,255));
barraVermelha.setBackground(new Color(255,0,0));
barraVermelha.addAdjustmentListener(this);
scrollBarPanel.add(barraVerde = new
      Scrollbar(Scrollbar.HORIZONTAL,0,0,0,255));
barraVerde.setBackground(new Color(0,255,0));
barraVerde.addAdjustmentListener(this);
scrollBarPanel.add(barraAzul = new
      Scrollbar(Scrollbar.HORIZONTAL,0,0,0,255));
barraAzul.setBackground(new Color(0,0,255));
barraAzul.addAdjustmentListener(this);
```

Exemplo: AdjustmentListener - cont.

```
        f.add("South", scrollBarPanel);
        f.setSize(400,300);
        f.setVisible(true);
    }
    public void adjustmentValueChanged(AdjustmentEvent e)
    {
        if (
             (e.getSource() == barraVermelha) ||
             (e.getSource() == barraVerde) ||
             (e.getSource() == barraAzul) )
        {
            canvas.setBackground( new Color(barraVermelha.getValue(),
                   barraVerde.getValue(),barraAzul.getValue()));
            infoLabel.setText("Color(" + barraVermelha.getValue() +
                   ”,” + barraVerde.getValue() + ",” +
                   barraAzul.getValue() + ")");
            canvas.repaint();
        }
    }
}
```

Como escrever um Component Listener

- Um ou mais eventos componentes são gerados quando o objeto componente que estava escondido torna-se: visível, ou é movido ou é redimensionado.
- O componente torna-se visível ou escondido através da invocação do método **setVisible**.
- A interface da **Component Listener** e a sua class adapter - **ComponentAdapter**, contém 4 métodos:

Os 4 métodos da ComponentListener

- void componentHidden(ComponentEvent)
  - ⇨é chamado pela AWT após o componente torna-se escondido (invisível), é resultado da invocação do método **setVisible(false)** para o objeto componente.
- void componentMoved(ComponentEvent)
  - ⇨é chamando pela AWT após a movimentação de um componente, relativo ao seu container. Por exemplo, se uma janela é movida, a janela gera um evento

Os 4 métodos da ComponentListener - cont.

- void componentResized(ComponentEvent)
  - ⇨é chamado pela AWT após ocorrer mudança na tamanho do componente (bordar retangular).
- void componentShow(ComponentEvent)
  - ⇨é chamado pela AWT após o componente torna-se visível como resultado da invocação do método **setVisible(true)** sobre o objeto.

Exemplo: ComponentListener

```
import java.awt.*;
import java.awt.event.*;
public class ComponentEventDemo implements
  ComponentListener
{
    private Frame tela;
    public static void main (String args[])
    {
       ComponentEventDemo c = new ComponentEventDemo();
       c.go();
    }
    public void go()
    {
       tela = new Frame("ComponentEvent");
       tela.setSize(300,300);
       tela.setBackground(Color.yellow);
       tela.setVisible(true);
```

Exemplo: ComponentListener

```
        tela.addComponentListener(this);
    }
    public void componentMoved(ComponentEvent e)
    {
        System.out.println("A janela moveu-se");
    }
    public void componentHidden(ComponentEvent e) {}
    public void componentResized(ComponentEvent e)
    {
        System.out.println ("A janela mudou o tamanho");
    }
    public void componentShown(ComponentEvent e) {}
}
```

Como escrever um ContainerListener

- Os eventos de container são gerados quando um componente é adicionado ou removido do container.
- A interface ContainerListener e sua classe adapter - ContainerAdapter possui dois métodos:
  - ⇨ void componentAdded (ContainerEvent)
    - é chamado pela AWT após um componente ser adicionado ao container.
  - ⇨ void componentRemoved(ContainerEvent)
    - é chamdo pela AWT após o componente ser removido do container.

Exemplo:ContainerListener

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class ContainerEventDemo implements
   ContainerListener,ActionListener
{
    TextArea display;
    Frame fr;
    Panel buttonPanel;
    Button addButton, removeButton, clearButton;
    Vector buttonList;
    static final String ADD    = "add";
    static final String REMOVE = "remove";
    static final String CLEAR  = "clear";
    String newline;
```

Exemplo:ContainerListener - cont.

```
public ContainerEventDemo()
{
  newline = System.getProperty ("line.separator");
  buttonList = new Vector(10,10);
  fr = new Frame("Teste de Container");
  addButton  = new Button("Adiciona um botao");
  addButton.setActionCommand(ADD);
  addButton.addActionListener(this);
  removeButton = new Button("Remove um botao");
  removeButton.setActionCommand(REMOVE);
  removeButton.addActionListener(this);
  buttonPanel = new Panel();
  buttonPanel.addContainerListener(this);
  display = new TextArea(5,20);
  display.setEditable(false);
  clearButton = new Button("Limpa a area de texto");
```

Exemplo:ContainerListener - cont.

```
clearButton.setActionCommand(CLEAR);
clearButton.addActionListener(this);

// lay out dos componentes

GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints c = new GridBagConstraints();
fr.setLayout (gridbag);
c.fill = GridBagConstraints.BOTH;
c.weighty = 1.0; // botão e mensagem de mesmo peso
c.gridwidth = GridBagConstraints.REMAINDER;
gridbag.setConstraints(display, c);
fr.add(display);
c.weighty = 0.0;
gridbag.setConstraints(clearButton,c);
fr.add(clearButton);
```

Exemplo:ContainerListener - cont.

```
    c.weightx = 1.0; // adicionar ou remove
                     // botões de igual peso
    c.gridwidth = 1;
    gridbag.setConstraints(addButton,c);
    fr.add(addButton);
    c.gridwidth = GridBagConstraints.REMAINDER;
    gridbag.setConstraints(removeButton, c);
    fr.add(removeButton);
    c.weighty = 1.0;
    gridbag.setConstraints(buttonPanel, c);
    fr.add(buttonPanel);
    fr.pack();
    fr.setVisible(true);
  }
  public void componentAdded(ContainerEvent e) {
    displayMessage(" adicionado a", e);
  }
```

Exemplo:ContainerListener - cont.

```
public void componentRemoved(ContainerEvent e) {
  displayMessage(" removido de ", e);
}
void displayMessage(String acao, ContainerEvent e) {
  display.append(((Button)e.getChild()).getLabel()
                 + " foi "
                 + acao
                 + e.getContainer().getClass().getName()
                 + newline);
}
public void actionPerformed (ActionEvent e) {
   String command = e.getActionCommand();
   if (command == ADD)
   {
      Button newButton = new Button("Botao #" +
                         (buttonList.size() +1));
```

Exemplo:ContainerListener - cont.

```
    buttonList.addElement(newButton);
    buttonPanel.add(newButton);
    buttonPanel.validate(); // faz o botao aparecer
}
else if (command == REMOVE)
{
    int lastIndex = buttonList.size() - 1;
    try {
        Button nixedButton =
          (Button)buttonList.elementAt(lastIndex);
          buttonPanel.remove (nixedButton);
          buttonList.removeElementAt(lastIndex);
          buttonPanel.validate(); // faz desaparecer
    } catch (ArrayIndexOutOfBoundsException exc) {}
}
else if (command == CLEAR)
```

Exemplo:ContainerListener - cont.

```
    display.setText("");
    }
    public static void main (String args[])
    {
       ContainerEventDemo d = new
            ContainerEventDemo();
    }
}
```

Como escrever FocusListener

- Vários componentes - mesmos aqueles primários que são manipulados pelo **mouse**, tais como o botões - podem ser manipulados pelo teclado. Para uma tecla pressionada afetar o componente, o componente deve ter um **focus keyboard**.
- Os eventos focus são gerados quando um componente ganha ou perde o foco do teclado.
- A forma como o componente ganha o foco depende do sistema de janelas. Tipicamente, o usuário fixa o foco em uma janela ou componente através do clique no mouse, pela tabulação entre componentes ou por outras formas de interagir com os componentes.

Os métodos do FocusListener

- A interface FocusListener e sua classe adapter - FocusAdapter, contém dois métodos:
  - void focusGained(FocusEvent)
    - é chamado pela AWT após o componente obter o foco.
  - void focusLost(FocusEvent)
    - é chamado pela AWT após o componente perder o foco.

Exemplo: FocusListener

```
import java.awt.*;
import java.awt.event.*;
public class FocusDemo implements FocusListener {
    private Frame tela;
    public static void main (String args[]) {
       FocusDemo c = new FocusDemo();
       c.go();
    }
    public void go()
    {
       tela = new Frame("FocusListener");
       tela.setSize(300,300);
       tela.setBackground(Color.green);
       tela.setVisible(true);
       tela.addFocusListener(this);
    }
```

Exemplo: FocusListener - cont.

```
public void focusGained(FocusEvent e) {
        System.out.println("O foco esta na
   janela");
     }
     public void focusLost(FocusEvent e) {
        System.out.println ("A janela perdeu o
   foco");
     }
}
```

Como escrever um KeyListener

- Os eventos chaves (key) informa quando o usuário está digitando no teclado. Especificamente, estes eventos são gerados pelos componentes que podem escutar eventos de foco.
- O usuário pode ser notificado em dois tipos básicos de eventos de chave:
  - ⇨ datilografando um caracter unicode (key typed);
  - ⇨ pressionando-se ou liberando-se uma chave no teclado (key pressed ou key released: F1, F2, ...) .

Os métodos da interface KeyListener

- void keyTyped (KeyEvent)
  - ⇨chamado pela AWT após ser digitado um caracter unicode em um componente que escuta o teclado.
- void keyPressed(KeyEvent)
  - ⇨chamado pela AWT após ser pressionado um chave enquanto o componente detém o foco.
- void keyReleased(KeyEvent)
  - ⇨chamado pela AWT após o usuário liberar a chave enquanto o componente detém o foco.

Exemplo: KeyListener

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

public class KeyEventDemo implements KeyListener,
   ActionListener {
     TextArea displayArea;
     TextField typingArea;
     String newline;
     Frame fr;
     public KeyEventDemo() {
         fr = new Frame("KeyEventDemo");
         Button button = new Button("Apagar");
         button.addActionListener(this);

         typingArea = new TextField(20);
         typingArea.addKeyListener(this);
```

Exemplo: KeyListener

```
    displayArea = new TextArea(5, 20);
    displayArea.setEditable(false);
    fr.setLayout(new BorderLayout());
    fr.add("Center", displayArea);
    fr.add("North", typingArea);
    fr.add("South", button);
    newline =       System.getProperty("line.separator");
    fr.pack();
    fr.setVisible(true);
}
/** Manipulando o evento de tecla digitada no campo
 texto */
public void keyTyped(KeyEvent e) {
    displayInfo(e, "Tecla digitada: ");
}
```

Exemplo: KeyListener

```
/** Manipulando o evento de chave pressionada no campo
  texto */
    public void keyPressed(KeyEvent e) {
        displayInfo(e, "Chave pressionada: ");
    }

    /** Manipulando o evento de chave liberada no campo
  texto */
    public void keyReleased(KeyEvent e) {
        displayInfo(e, "Chave liberada: ");
    }

    /** Manipulando o clique no botao */
    public void actionPerformed(ActionEvent e) {
        //Apagando o componente texto
        displayArea.setText("");
        typingArea.setText("");
```

Exemplo: KeyListener

```
    //Retornando o foco na area digitada
     typingArea.requestFocus();
  }
  protected void displayInfo(KeyEvent e, String s){
     String charString, keyCodeString, modString,
tmpString;

     char c         = e.getKeyChar();
     int keyCode    = e.getKeyCode();
     int modifiers = e.getModifiers();

     if (Character.isISOControl(c)) {
         charString = "Caracter chave = (um caracter de
controle nao imprimivel)";
     } else {
         charString = "Caracter chave = '" + c + "'";
     }
```

Exemplo: KeyListener

```
        keyCodeString = "Codigo da chave = " + keyCode
                        + " ("
                        + KeyEvent.getKeyText(keyCode)
                        + ")";
        modString = "modificador = " + modifiers;
        tmpString =
KeyEvent.getKeyModifiersText(modifiers);
        if (tmpString.length() > 0) {
            modString += " (" + tmpString + ")";
        } else {
            modString += " (nao eh um modificador)";
        }
        displayArea.append(s
                           + newline + "    "
                           + charString
                           + newline + "    "
                           + keyCodeString
```

Exemplo: KeyListener

```
                                + newline + "    "
                                + modString
                                + newline);
    }
    public static void main(String args[])
    {
        KeyEventDemo k= new KeyEventDemo();
    }
}
```

A classe KeyEvent

- Esta classe define os seguintes métodos úteis:
  - int getKeyChar()
  - void setKeyChar(int)
    - pega ou atribui o caracter unicode associado com este evento.
  - int getKeyCode()
  - void setKeyCode(int)
    - pega ou atribui o código chave associado com este evento.

A classe KeyEvent

- void setModifiers(int)
  - atribui o estado da chave modificadora para este evento. Pode-se pegar a chave modificadora através do método getModifiers da classe InputEvent
- String getKeyText()
- String getKeyModifiersText()
  - retorna a descrição do texto da chave do evento e da chave modificadora.

Como escrever MouseListener

- Este evento informa quando o usuário usa o mouse para interagir com o componente.
- Ele ocorre quando o cursor entra ou sai de um componente na área da tela e o usuário pressiona ou libera o botão do mouse.
- A interface MouseListener e a classe MouseAdapter contém os métodos a seguir.
  - void mouseClicked(MouseEvent)
    - ocorre quando o click do mouse é pressionado sobre um componente.
  - void mouseEntered(MouseEvent)
    - ocorre assim que o cursor entra nos limites do componente.

Os métodos da MouseListener - cont.

- void mouseExited (MouseEvent)
  - ocorre assim que o cursor sai dos limites do componente.
- void mousePressed(MouseEvent)
  - ocorre assim que o usuário pressiona o botão do mouse enquanto o cursor está sobre o componente.
- void mouseReleased(MouseEvent)
  - ocorre quando o usuário libera o botão do mouse que foi pressionado sobre o componente.

A classe MouseEvent

- Os métodos da interface MouseListener e MouseAdapter possui um único parametro: um objeto MouseEvent. A class MouseEvent define os seguintes métodos úteis:
  - int getClickCount()
    - retorna o número de click rápidos e consecutivos que o usuário tenha feito.
  - int getx(), int gety() e int getPoint()
    - retorna as coordenadas (x,y) da posição que ocorreu o evento, relativo ao componente que gerou o evento.
  - boolean isPopupTrigger()
    - retorna true se o click no mouse fez o menu popup aparecer.

Como escrever MouseMotionListener

- O MouseMotionListener informa quando usuário usou mouse para move-se sobre a tela. E possui os seguintes métodos:
  - void mouseDragged(MouseEvent)
    - é chamado pela AWT assim que o usuário movimenta o mouse mantendo-se o botão do mouse pressionado. Este evento é disparado pelo componente que gerou o mais recente evento de mousePressed, mesmo o cursor não esteja sobre o componente.
  - void mouseMoved(MouseEvent)
    - é chamado pela AWT em resposta a movimentação do mouse, pelo usuário, sem que o botão do mouse esteja pressionado.

Exemplo:MouseMotionListener

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionEventDemo implements
  MouseMotionListener {
    BlankArea blankArea;
    TextArea textArea;
    static final int maxInt =
  java.lang.Integer.MAX_VALUE;
    String newline;
    Frame fr;
    public MouseMotionEventDemo() {
        fr = new Frame();
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new
                  GridBagConstraints();
        fr.setLayout(gridbag);
        c.fill = GridBagConstraints.BOTH;
```

Exemplo:MouseMotionListener

```
c.gridwidth = GridBagConstraints.REMAINDER;
c.weightx = 1.0;
c.weighty = 1.0;

c.insets = new Insets(1, 1, 1, 1);
blankArea = new BlankArea(new Color
          (0.98f, 0.97f, 0.85f));
gridbag.setConstraints(blankArea, c);
fr.add(blankArea);

c.insets = new Insets(0, 0, 0, 0);
textArea = new TextArea(5, 20);
textArea.setEditable(false);
gridbag.setConstraints(textArea, c);
fr.add(textArea);
```

Exemplo:MouseMotionListener

```
//Registra o evento de mouse na blankArea e panel
        blankArea.addMouseMotionListener(this);
        fr.addMouseMotionListener(this);

        newline = System.getProperty
                            ("line.separator");
        fr.pack();
        fr.setVisible(true);
    }

    public void mouseMoved(MouseEvent e) {
       saySomething("Mouse movido; # de cliques: "
                    + e.getClickCount(), e);
    }
```

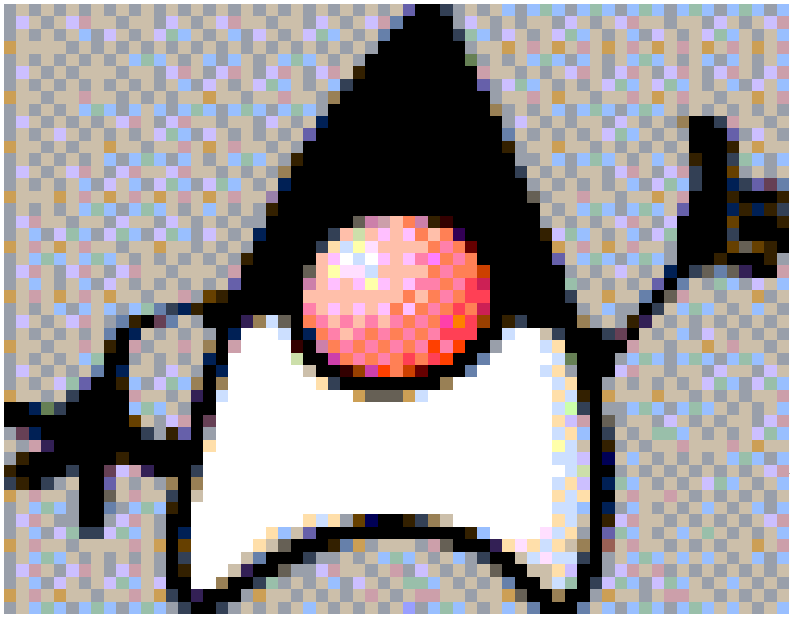
Exemplo:MouseMotionListener

```
public void mouseDragged(MouseEvent e) {
        saySomething("Mouse com botao pressionado; # de
                cliques: " + e.getClickCount(), e);
    }

    void saySomething(String eventDescription, MouseEvent e)
   {
        textArea.append(eventDescription +
                " detectado sobre " + e.getComponent().
                getClass().getName()+ newline);
        textArea.setCaretPosition(maxInt);
    }
    public static void main (String args[])
    {
        MouseMotionEventDemo m= new MouseMotionEventDemo();
    }
}
```

Como escrever TextListener

- Eventos desse tipo são gerados assim que o texto de um componente text tenha de alguma maneira mudado e possui um único método:
  - void textValueChange(TextEvent)
    - é chamado pela AWT assim que o texto em uma componente tenha mudado.
- Ver o exemplo texto em ActionListener.

Applet



Applet - ordem de inicialização

- init: executado quando a applet é carregada ou recarregada (menos para Netscape).
- start: executado após o método init ou quando a página é recarregada.
- stop: executado quando o usuário muda de página, continuando na memória, ou quando a execução do browser é encerrada.
- destroy: executada quando a applet for ser removida da memória.

Applet - exemplo

```
import java.applet.*;
import java.awt.*;
import java.awt.Graphics;
public class TesteApplet extends Applet
{
   StringBuffer mensagem;
   int cont;
   public void init() {
      mensagem = new StringBuffer();
      adicionaMsg("entrou no init() "); }
   public void start() {
      cont = cont + 1;
      adicionaMsg("entrou no start() " + cont
       + "  ");
   } //Continua …
```

Applet - exemplo

```
    public void stop() {
       adicionaMsg("entrou no stop() ");  }
    public void adicionaMsg(String msg)  {
       System.out.println(msg);
       mensagem.append(msg);
       repaint();   }
    public void paint(Graphics g)   {
       g.drawString(mensagem.toString(), 5,15);
    }
} //Fim
```

Applet e HTML

< applet codebase = URL

(opcional)

code = arquivoExecutável (obrigatório)

width = pixelLargura (obrigatório)

height = pixelAltura >

(obrigatório)

< param name = nome value = valor >

. . .

</applet>

Applet e HTML

No código HTML:

```
< param name=parTipo  value=intranet >
< param name=parValor  value=50 >
```

No código da Applet:

```
String tipo = getParameter(“parTipo”);
int valor = (new Integer (getParameter("parValor")))
            .intValue();
```

Applet e URL

```
import java.net.URL;
.  .  .
getAppletContext().showDocument(new URL(url, opção)
```

onde opção:

- `_self`: mostrar no frame atual
- `_parent`: mostrar no frame pai
- `_top`: mostrar no primeiro frame
- `_blank`: mostrar em uma nova janela do browser

Bibliografia

- Cornell, Gary. Core Java; tradução Daniel Vieira; revisão técnica Rodrigo Rodrigues. São Paulo, Makron Books, 1997.
- Eckel, Bruce. Thinking in Java. Prentice Hall PTR, USA, 1998.
- Campione, Mary. The Java Tutorial: object-oriented programming for the internet. Addison Wesley Longman, California- USA, 1998.
- Deitel, H. M.;Deitel, P. J. Java, Como Programar. 3ª. edição – Porto Alegre: Bookman, 2001.
- Java Tutorial : http://java.sun.com/docs/books/tutorial/java/TOC.html#nutsandbolts.
- JAVA Swing (JFC) : http://java.sun.com/docs/books/tutorial/uiswing/TOC.html#start.
- JAVADOC: http://java.sun.com/j2se/1.4/docs/api/index.html.
- Classes Essenciais Java: http://java.sun.com/docs/books/tutorial/essential/TOC.html.
- JAVA WORLD e CORE JAVA: http://www.javaworld.com/channel_content/jw-core-index.shtml.