CURSO: Ciência da Computação	**DATA: ____/____/ 2013**
PERÍODO: 4º.	**PROFESSOR: Andrey**
DISCIPLINA: Técnicas Alternativas de Programação	**AULA: 08**

APRESENTAÇÃO

Na aula de hoje vamos apresentar e discutir conceitos relacionados com a implementação de interfaces gráficas em java.

DESENVOLVIMENTO

As interfaces gráficas em Java podem ser de três tipos:
Applets;
Aplicativos (Pacotes awt, Swing)
Web (JSP e Servlets)

Diálogos

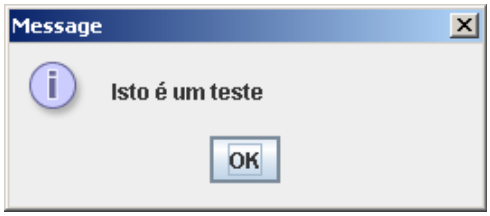
Os Diálogos são janelas simples e padronizadas que servem para realizar um diálogo com o usuário. Elas são criadas e logo que o usuário dá a resposta elas são fechadas e o resultado retorna para o programa. Os Diálogos usam a classe JOptionPane.

Os principais tipos de diálogos são:

ConfirmDialog	Pede confirmação para o usuário sobre uma ação
InputDialog	Permite a entrada de texto pelo usuário
MessageDialog	Exibe uma mensagem para o usuário
OptionDialog	Diálogo que pode realizar todas as formas anteriores

Exemplo:

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

public class Dialogo
{
    public static void main(String[] args)
    {
        String frase = JOptionPane.showInputDialog(null,
                       "digite a frase", "janela1",0);
        JOptionPane.showMessageDialog(null, frase);
    }
}
```



Frames.

Um Frame é uma classe que permite a construção de janelas. O Frame gera uma janela com bordas e uma barra de título. Um Frame se diferencia de um Applet principalmente pela forma que ele é executado. Enquanto um Applet necessita do Browser (ou appletviewer) para ser executado, um Frame é independente, e pode ser executado como se fosse uma aplicação normal.

Os componentes de um Frame são basicamente os mesmos que de um Applet. A grande diferença é que os componentes são definidos e colocados no método construtor da classe. Além disso existe a necessidade de se colocar um método principal quando se trata da janela inicial de uma aplicação.

Exemplo de um frame:


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

public class ExemploFrame01 extends JFrame implements ActionListener
{
	private JButton B1,B2;
	private JLabel L1, L2;
	private JTextField T1,T2;
	public ExemploFrame01()

	{
		B1 = new JButton("Botao 1");
		B2 = new JButton("Botao 2");
		B1.addActionListener(this);
		B2.addActionListener(this);
		L1 = new JLabel("Label 1");
		L2 = new JLabel("Label 2");
		T1 = new JTextField("TextField 1");
		T2 = new JTextField("TextField 2");
		this.getContentPane().setLayout(new FlowLayout());
		this.getContentPane().add(L1);
		this.getContentPane().add(T1);
		this.getContentPane().add(L2);
		this.getContentPane().add(T2);
		this.getContentPane().add(B1);
		this.getContentPane().add(B2);
		this.setLocation(200,200);
		this.setSize(200,150);
	}

	public void actionPerformed(ActionEvent e)
	{
		//aqui vai o código para tratar os eventos dos botões
		if (e.getSource() == B1)
		{
```

```
                System.out.println(”voce pressionou B1”+T1.getText());
            }
            if (e.getSource() == B2)
            {
                System.out.println(“voce pressionou B2”+T2.getText());
            }
        }

        public static void main(String args[])
        {
            JFrame janela = new     ExemploFrame01();
            janela.show();
            WindowListener x = new WindowAdapter ()
            {
                public void windowClosing(WindowEvent e)
                {
                    System.exit(0);
                }
            };
            janela.addWindowListener(x);

        }
}
```

Componentes de Interface Gráfica

Button

São componentes de interface que representam um botão que possui a metáfora de um botão de um aparelho elétrico. O botão se comporta da seguinte maneira: quando não está selecionado ou quando o cursor do mouse não está sobre ele a aparência é do desenho a esquerda; quando o botão está selecionado a aparência é a do desenho do meio; quando ele é pressionado a aparência é a da direita.

CheckBox

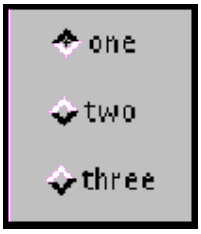
São itens de interface que podem ser "marcados" ou "desmarcados" podendo estar associados a um valor verdadeiro ou falso. Cada checkbox pode ser usado de maneira independente ou em grupo.
De maneira independente:

```
setLayout(new GridLayout(3, 1));
add(new Checkbox("one", null, true));
add(new Checkbox("two"));
add(new Checkbox("three"));
```

CheckBoxGroup

São Grupos de CheckBox. Quando os CheckBox são usados em grupo, apenas um deles pode estar "marcado" em um determinado momento. Quando um é "marcado" os outros vão para o estado "desmarcado". O CheckBoxGroup usa a metáfora de botões de rádios antigos.

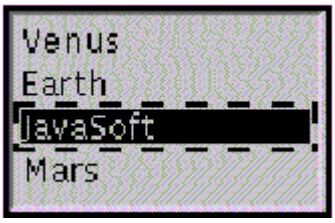
```
setLayout(new GridLayout(3, 1));
CheckboxGroup cbg = new CheckboxGroup();
add(new Checkbox("one", cbg, true));
add(new Checkbox("two", cbg, false));
add(new Checkbox("three", cbg, false));
```



Lists

São listas de itens que podem ser marcados ou desmarcados. A lista pode ser rolável

```
List lst = new List(4, false);
lst.add("Mercury");
lst.add("Venus");
lst.add("Earth");
lst.add("JavaSoft");
lst.add("Mars");
cnt.add(lst);
```



ScrollBar

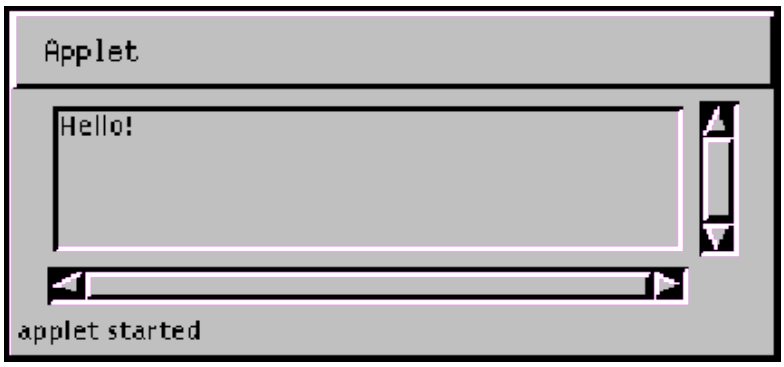
É uma barra que permite a rolagem de um determinado elemento.

```
ranger = new Scrollbar(Scrollbar.HORIZONTAL, 0, 60, 0, 300);
add(ranger);
```

TextArea

São campos de texto que permitem a entrada e a edição de textos com mais de uma linha

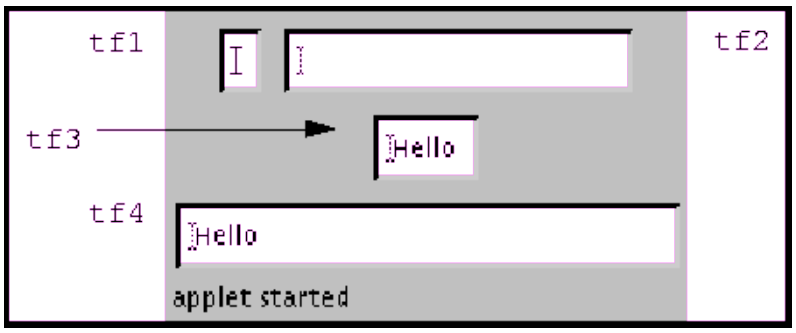
```
new TextArea("Hello", 5, 40);
```



TextField

Campo que permite a inserção ou edição de texto com uma linha.

```
TextField tf1, tf2, tf3, tf4;
tf1 = new TextField();
tf2 = new TextField("", 20);
tf3 = new TextField("Hello!");
tf4 = new TextField("Hello", 30);
```



Label

Permite exibir mensagens ou texto dentro de uma janela. Este campo não permite a alteração do texto.

```
setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
add(new Label("Hi There!"));
add(new Label("Another Label"));
```

Gerenciadores de Layout

São elementos das telas que servem para gerenciar a disposição dos componentes da interface.

FlowLayout

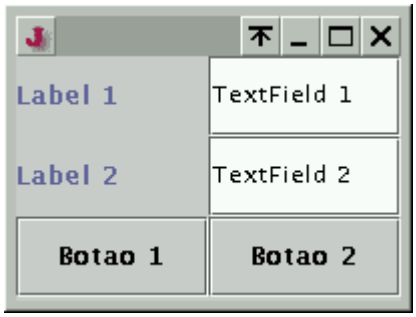
É o mais simples dos gerenciadores de Layout. No FlowLayout os componentes são dispostos da esquerda para a direita como se fosse uma linha. Quando o espaço horizontal não é suficiente uma outra linha é criada.

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

public class ExemploFrame01 extends JFrame implements ActionListener
{
      private JButton B1,B2;
      private JLabel L1, L2;
      private JTextField T1,T2;
      public ExemploFrame01()

      {
            B1 = new JButton("Botao 1");
            B2 = new JButton("Botao 2");
            B1.addActionListener(this);
            B2.addActionListener(this);
            L1 = new JLabel("Label 1");
            L2 = new JLabel("Label 2");
            T1 = new JTextField("TextField 1");
            T2 = new JTextField("TextField 2");

            // trecho de código a ser substituído nos outros exemplos

            this.getContentPane().setLayout(new FlowLayout());
            this.getContentPane().add(L1);
            this.getContentPane().add(T1);
            this.getContentPane().add(L2);
            this.getContentPane().add(T2);
            this.getContentPane().add(B1);
            this.getContentPane().add(B2);

            // final do trecho de código a ser substituído nos outros exemplos

            this.setLocation(200,200);
            this.setSize(200,150);
      }

      public void actionPerformed(ActionEvent e)
      {
            //aqui vai o código para tratar os eventos dos botões
      }

      public static void main(String args[])
      {
            JFrame janela = new     ExemploFrame01();
            janela.show();
            WindowListener x = new WindowAdapter ()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        System.exit(0);
                  }
            };
            janela.addWindowListener(x);

      }
}
```

GridLayout

É o gerenciador que divide o espaço da janela em células espalhadas em uma grade retangular. Todas as células terão o mesmo tamanho. O posicionamento dos componentes depende da ordem em que eles são adicionados à janela.

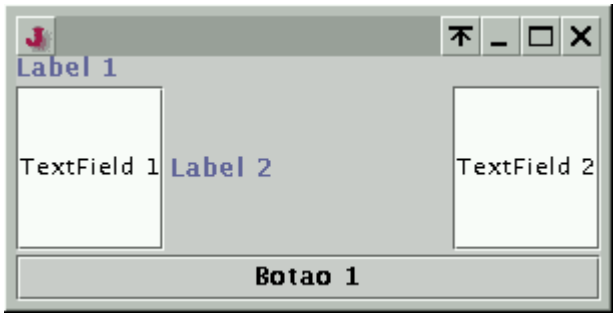
```
this.getContentPane().setLayout(new GridLayout(3,2));
this.getContentPane().add(L1);
this.getContentPane().add(T1);
this.getContentPane().add(L2);
this.getContentPane().add(T2);
this.getContentPane().add(B1);
this.getContentPane().add(B2);
```



BorderLayout

Divide o espaço da janela em 5 regiões: North, West, Center, East, South. O North é a região superior da janela, o West é a região à esquerda, o Center é a região central, o East é a região à direita e o South é a região. Cada componente é adicionado a uma região específica. Se houver necessidade de se colocar mais de um componente em uma mesma região será preciso adicioná-los em um Panel (visto a seguir).

```
this.getContentPane().setLayout(new BorderLayout(3,2));
this.getContentPane().add("North",L1);
this.getContentPane().add("West",T1);
this.getContentPane().add("Center",L2);
this.getContentPane().add("East",T2);
this.getContentPane().add("South",B1);
```

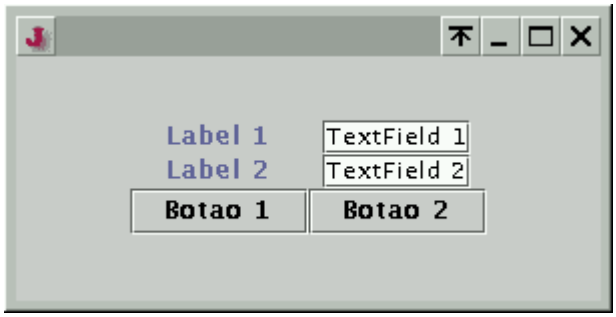


GridBagLayout

É um gerenciador mais aprimorado, também baseado em uma grade, mas que permite que seja configurado o tamanho de cada linha ou coluna. Além disso, cada elemento é colocado na sua respectiva célula.

```
GridBagLayout gridbag = new GridBagLayout();
```

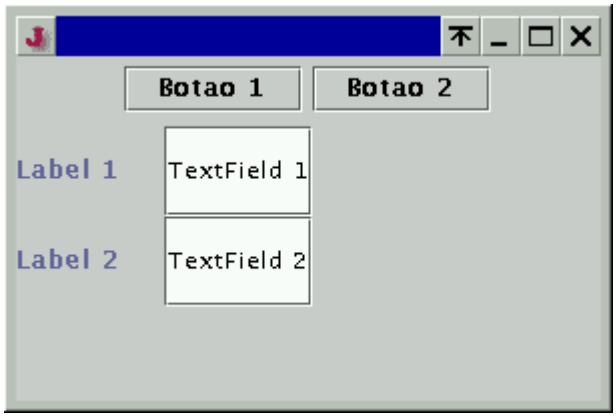
```
GridBagConstraints c = new GridBagConstraints();
this.getContentPane().setLayout(gridbag);
c.gridy = 0;
c.gridx = 0;
gridbag.setConstraints(L1, c);
this.getContentPane().add(L1);
c.gridy = 0;
c.gridx= 1;
gridbag.setConstraints(T1, c);
this.getContentPane().add(T1);
c.gridy = 1;
c.gridx= 0;
gridbag.setConstraints(L2, c);
this.getContentPane().add(L2);
c.gridy = 1;
c.gridx= 1;
gridbag.setConstraints(T2, c);
this.getContentPane().add(T2);
c.gridy = 2;
c.gridx= 0;
gridbag.setConstraints(B1, c);
this.getContentPane().add(B1);
c.gridy = 2;
c.gridx= 1;
gridbag.setConstraints(B2, c);
this.getContentPane().add(B2);
```



Panel

São elemento que servem para agrupar outros elementos da interface. Quando se deseja colocar mais de um componente, é necessário que eles sejam colocados em um Panel. Eles funcionam como se fossem divisões das janelas.

```
this.getContentPane().setLayout(new BorderLayout(3,2));

JP1 = new JPanel();

JP1.setLayout(new FlowLayout());
JP1.add(B1);
JP1.add(B2);

JP2 = new JPanel();

JP2.setLayout(new GridLayout(3,2));
JP2.add(L1);
JP2.add(T1);
JP2.add(L2);
JP2.add(T2);

this.getContentPane().add("North",JP1);
this.getContentPane().add("West",JP2);
```



Tratamento de Eventos de Interface

Eventos de Mouse

Os eventos de teclado são tratados através das interfaces **MouseListener e MouseMotionListener**

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

public class MouseTracker extends JFrame
                          implements MouseListener, MouseMotionListener
{
   private JLabel statusBar;

   public MouseTracker()
   {
      super( "Demonstrating Mouse Events" );
      statusBar = new JLabel();
      getContentPane().add( statusBar, BorderLayout.SOUTH );
      // application listens to its own mouse events
      addMouseListener( this );
      addMouseMotionListener( this );
      setSize( 275, 100 );
      show();
   }

   // MouseListener event handlers
   public void mouseClicked( MouseEvent e )
   {  statusBar.setText( "Clicked at [" + e.getX() + ", " + e.getY() + "]" ); }

   public void mousePressed( MouseEvent e )
   {  statusBar.setText( "Pressed at [" + e.getX() + ", " + e.getY() + "]" ); }

   public void mouseReleased( MouseEvent e )
   {  statusBar.setText( "Released at [" + e.getX() + ", " + e.getY() + "]" ); }

   public void mouseEntered( MouseEvent e )
   {  statusBar.setText( "Mouse in window" ); }

   public void mouseExited( MouseEvent e )
   {  statusBar.setText( "Mouse outside window" );  }
```

```
   // MouseMotionListener event handlers
   public void mouseDragged( MouseEvent e )
   {  statusBar.setText( "Dragged at [" + e.getX() + ", " + e.getY() + "]" ); }

   public void mouseMoved( MouseEvent e )
   {  statusBar.setText( "Moved at [" + e.getX() + ", " + e.getY() + "]" ); }

   public static void main( String args[] )
   {
      MouseTracker app = new MouseTracker();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```

Tratamento de Eventos do Teclado

Os eventos de teclado são tratados através da interface KeyListener

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

public class KeyDemo extends JFrame implements KeyListener {
   private String line1 = "", line2 = "";
   private String line3 = "";
   private JTextArea textArea;

   public KeyDemo()
   {
      super( "Demonstrating Keystroke Events" );

      textArea = new JTextArea( 10, 15 );
      textArea.setText( "Press any key on the keyboard..." );
      textArea.setEnabled( false );

      // allow frame to process Key events
      addKeyListener( this );

      getContentPane().add( textArea );

      setSize( 350, 100 );
      show();
   }

   public void keyPressed( KeyEvent e )
   {
      line1 = "Key pressed: " + e.getKeyText( e.getKeyCode() );
      setLines2and3( e );
   }

   public void keyReleased( KeyEvent e )
   {
```

```
      line1 = "Key released: " + e.getKeyText( e.getKeyCode() );
      setLines2and3( e );
   }

   public void keyTyped( KeyEvent e )
   {
      line1 = "Key typed: " + e.getKeyChar();
      setLines2and3( e );
   }

   private void setLines2and3( KeyEvent e )
   {
      line2 = "This key is " +
              ( e.isActionKey() ? "" : "not " ) +
              "an action key";

      String temp =
         e.getKeyModifiersText( e.getModifiers() );

      line3 = "Modifier keys pressed: " +
              ( temp.equals( "" ) ? "none" : temp );

      textArea.setText(
         line1 + "\n" + line2 + "\n" + line3 + "\n" );
   }

   public static void main( String args[] )
   {
      KeyDemo app = new KeyDemo();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```

Classes Adaptadoras

Para Tratarmos eventos podemos usar classes adaptadoras. As Classes Adaptadoras são classes que implementam as classes de tratamento de eventos. Para usar as classes adaptadoras podemos criar uma nova classe adaptadora, que herda de uma classe adaptadora do Java. Isto pode ser feito criando uma classe interna ou fazendo a declaração on-line de uma classe.

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

public class Painter extends JFrame {
   private int xValue = -10, yValue = -10;

   public Painter()
   {
      super( "A simple paint program" );
```

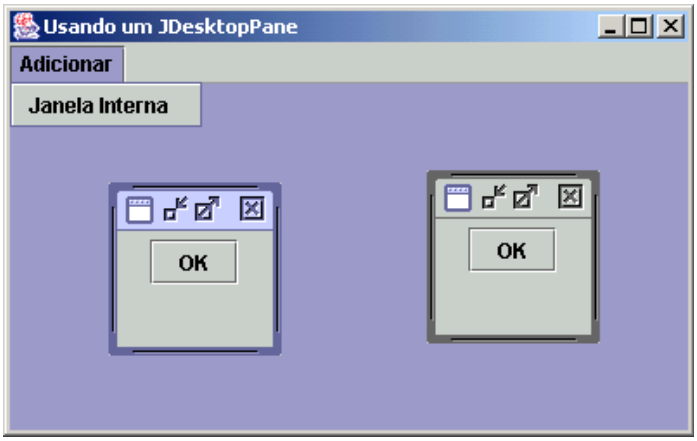
```
      getContentPane().add(
         new Label( "Drag the mouse to draw" ),
         BorderLayout.SOUTH );

      addMouseMotionListener(new MouseMotionHandler ());
      setSize( 300, 150 );
      show();
   }

   public void paint( Graphics g )
   {
      g.fillOval( xValue, yValue, 4, 4 );
   }

   public static void main( String args[] )
   {
      Painter app = new Painter();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }

   private class MouseMotionHandler extends MouseMotionAdapter
   {
            public void mouseDragged( MouseEvent e )
            {
               xValue = e.getX();
               yValue = e.getY();
               repaint();
            }
    }
}
```

ou substituindo a declaração de uma classe interna.

```
      addMouseMotionListener(
         new MouseMotionAdapter() {
            public void mouseDragged( MouseEvent e )
            {
               xValue = e.getX();
               yValue = e.getY();
               repaint();
            }
         }
      );
```

Janelas Internas

O programa abaixo irá criar uma janela como esta. Com um menu Adicionar e um item de menu Janela Interna. A Janela principal possui um espaço interno chamado JDeskTopFrame no qual é possível inserir janelas internas que são objetos da classe JInternalFrame.



Classe TesteMenu

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

public class TesteMenu extends JFrame implements ActionListener
{
    private JMenuBar barra;
    private JMenu menu;
    private JMenuItem item;
    private JDesktopPane desktop;

    public TesteMenu()
    {
        super( "Usando um JDesktopPane" );
        this.barra = new JMenuBar();
        this.menu = new JMenu( "Adicionar" );
        this.item = new JMenuItem( "Janela Interna" );
        this.menu.add( this.item );
        this.barra.add( this.menu );
        this.setJMenuBar( this.barra );
        this.desktop = new JDesktopPane();
        this.getContentPane().add( this.desktop );
        item.addActionListener(this);
        this.setSize( 500, 400 );
        this.setVisible(true);
   }

   public static void main( String args[] )
   {
      TesteMenu app = new TesteMenu();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }

   public void actionPerformed(ActionEvent e)
   {
       JanelaInterna frame = new JanelaInterna();
```

```
        this.desktop.add( frame );
    }

}
```

Classe JanelaInterna

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

class JanelaInterna extends JInternalFrame implements ActionListener
{
    private JButton B1;
    private JPanel panel;

    public JanelaInterna()
    {
        super("Janela Interna", true, true, true, true );
        panel = new JPanel();
        this.getContentPane().add(panel, BorderLayout.CENTER );
        this.setSize(100,100);
        this.setVisible(true);
        this.B1 = new JButton("OK");
        panel.setLayout(new FlowLayout());
        panel.add(B1);
        this.B1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource()== this.B1)
            System.out.println("apertou B1");
    }
}
```

ATIVIDADE

a) Crie uma janela para o cadastramento de um produto com os seguintes dados: nome, preço, quantidade em estoque e linha (móvel/eletrodoméstico).

b) Crie uma janela para cadastrar um vendedor com os dados: nome, salário fixo, percentual de comissão.

c) Crie uma classe Vendedor para manipular os dados do vendedor.

d) Crie uma janela para o cadastramento de um Produto, e uma classe para manipular os dados do produto.

e) Crie uma janela com um menu e as opções Produto e Vendedor e adapte as Janelas de cadastro de Produto e Vendedor para serem abertas na parte interna da janela principal.

BIBLIOGRAFIA BÁSICA

DEITEL, H. M. e DEITEL, P. J.*. Java, como Programar.* Ed. Bookman. Porto Alegre. 2001.

AHO, A.V.; HOPCROFT, J.E.; ULLMAN, J.D. Data Structures and Algorithms. Addison-Wesley, Reading, Massachusetts, 1983.