Graphical user Interfaces

Objetivos:

. Construir programas com interfaces gráficas

"Graphical User Interface (GUI) application programs"

- Utilizar as classes **JFrame, JButton, JLabel**, **ImageIcon**, **JTextField**, **JTextArea** e **JMenu** do package **javax.swing**

- Usar um modelo de programação por eventos

("Java's delegation-based event model")

Os componentes gráficos da linguagem Java estão definidos nos packages:

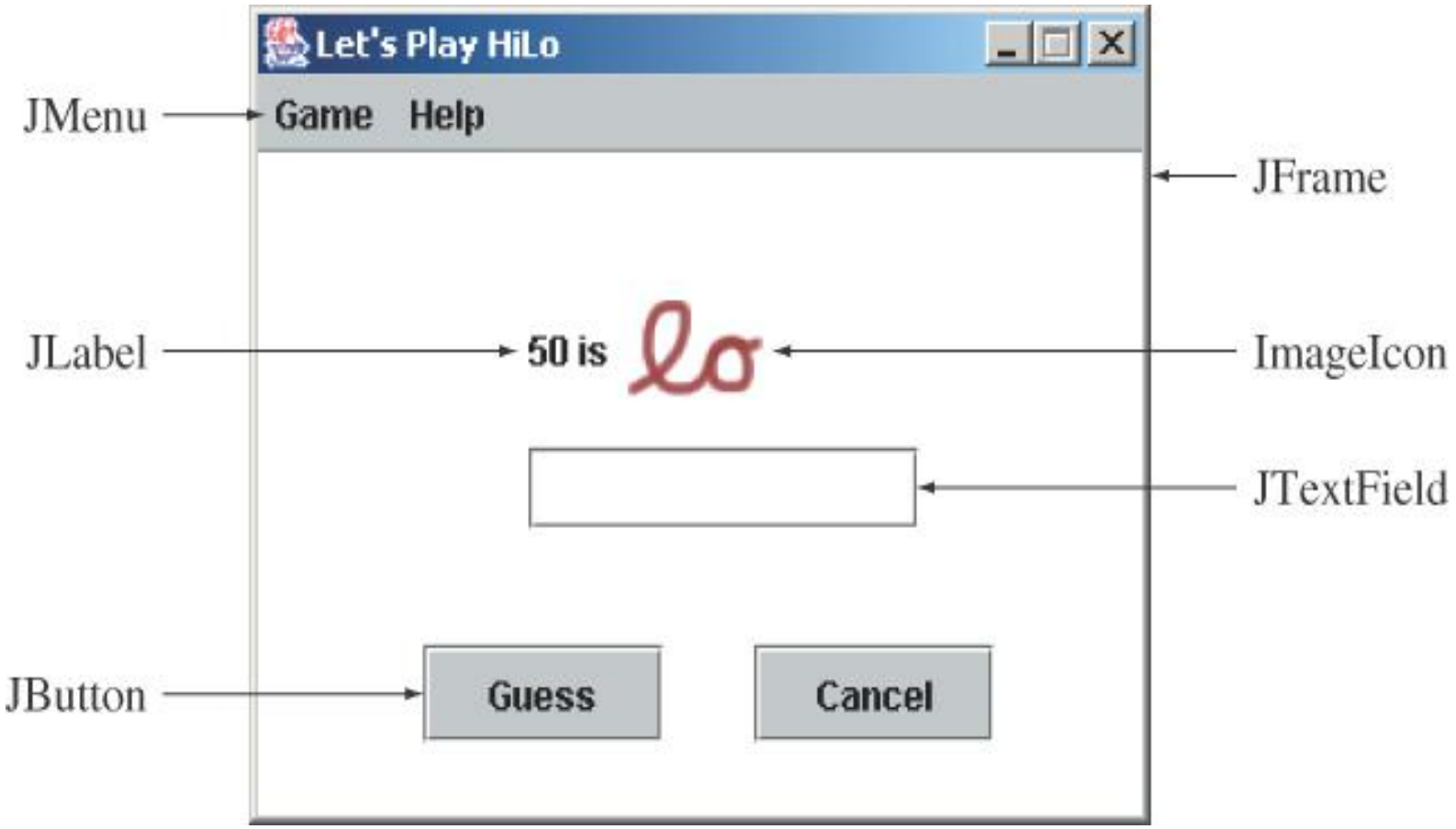
java.awt e

javax.swing (surge com a versão Java2 SDK1.2)

As classes Swing permitem maior portabilidade entre diferentes sistemas operativos.

(totalmente implementadas em Java)

Alguns componentes gráficos do package javax.swing



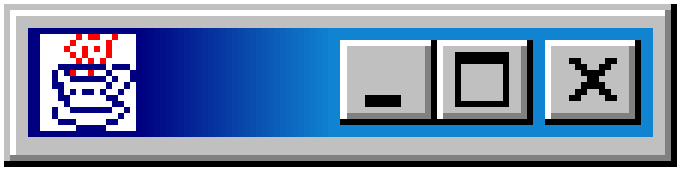
Definição de uma janela (JFrame)

- Um objeto do tipo JFrame contém os elementos básicos para manipularmos uma janela:

abrir, fechar, mover e redimensionar

- A uma janela poderemos adicionar outros componentes gráficos

```
import javax.swing.*;
public class Janela {
  public static void main (String [] args){
     JFrame janela = new JFrame();
     janela.setVisible(true);
  } }
```



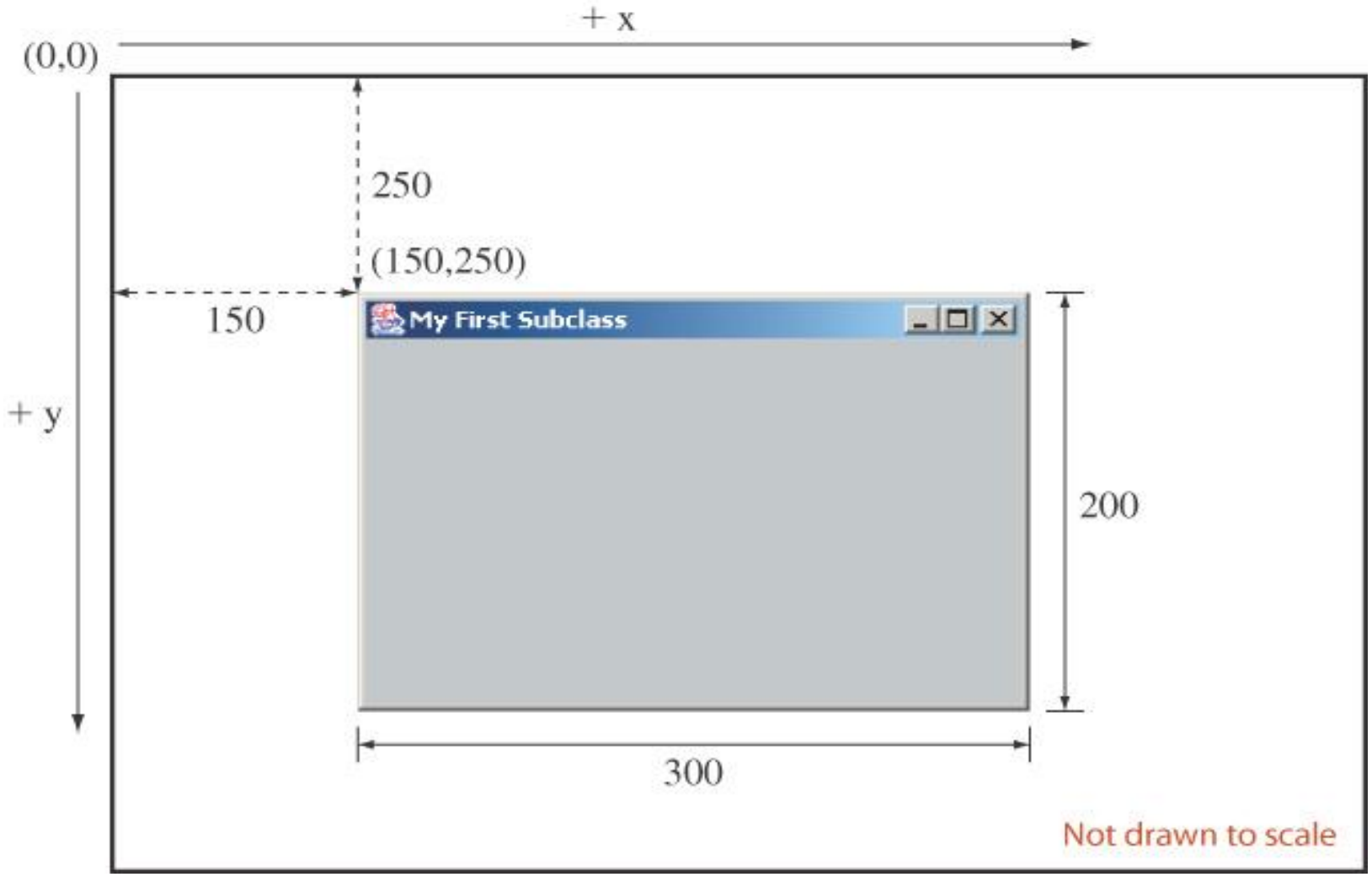
Alguns métodos da classe JFrame:

- atribuir um título
setTitle (“My First Subclass");

-dimensionar a janela
setSize (300, 200);
(300 pixels de largura e 200 pixels de altura)

- posicionar a janela no ponto de coordenadas (150, 250)
setLocation (150, 250);

- requerer que o programa termine quando a janela é fechada
setDefaultCloseOperation (EXIT_ON_CLOSE);


+ x
(0,0)
250
(150,250)
150
My First Subclass
+ y
200
300
Not drawn to scale

Para construir uma interface gráfica podemos definir uma subclasse de JFrame.

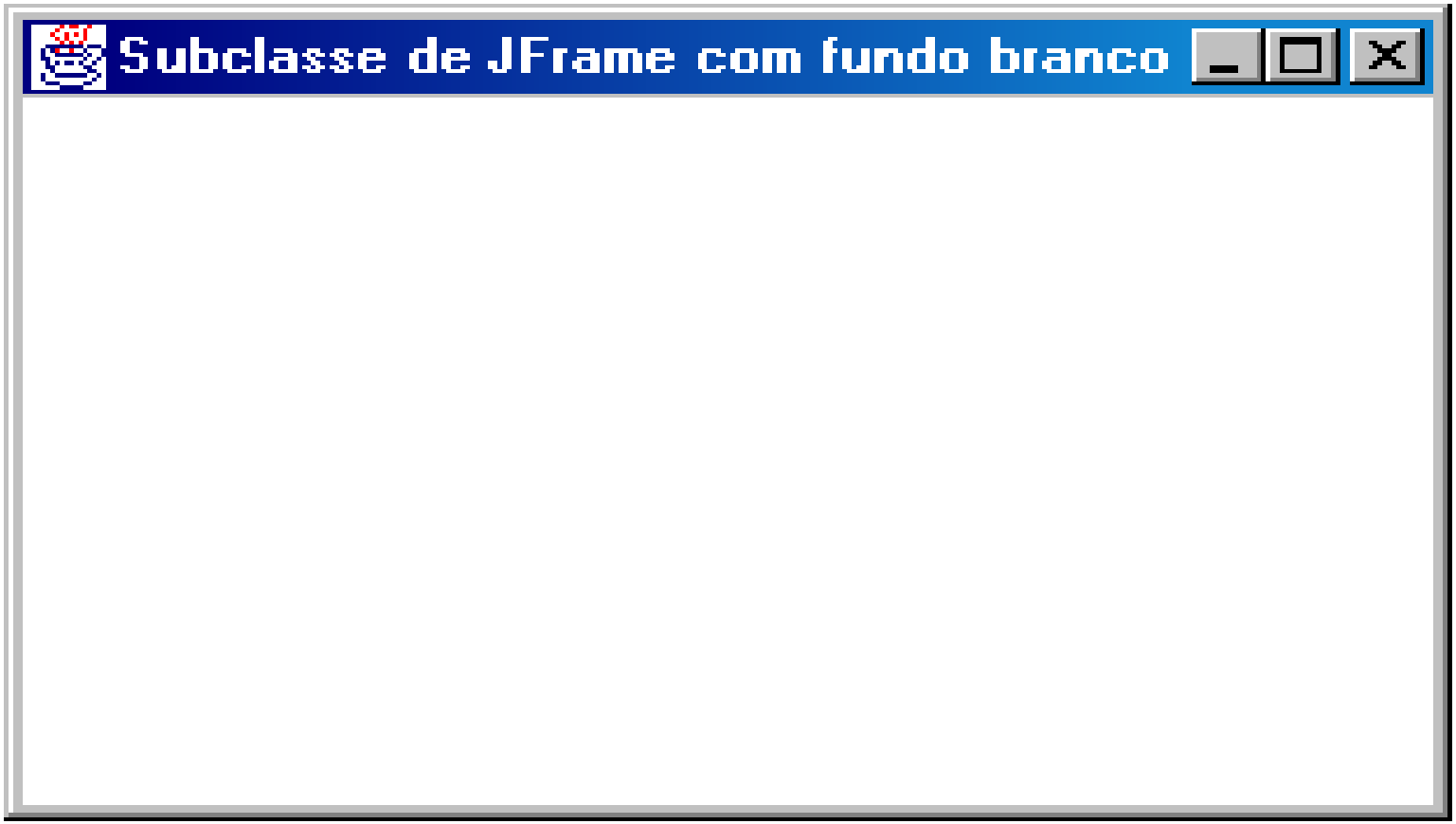
Nessa subclasse iremos adicionar o comportamento necessário à interface que pretendemos.

A classe Janela1 define uma janela com as características anteriores e com o fundo branco:

```
import javax.swing.*;
import java.awt.*;
public class Janela1  extends JFrame {
private static final int FRAME_WIDTH    = 300;
private static final int FRAME_HEIGHT   = 200;
private static final int FRAME_X_ORIGIN = 150;
private static final int FRAME_Y_ORIGIN = 250;
```

```
…
public Janela1(){
setTitle ( “Subclasse de JFrame com fundo branco" );
setSize  (FRAME_WIDTH, FRAME_HEIGHT );
setLocation (FRAME_X_ORIGIN, FRAME_Y_ORIGIN );
setDefaultCloseOperation( EXIT_ON_CLOSE );
alterarCorFundo( );
}
//continua …
```

```
private void alterarCorFundo() {
    Container contentor = getContentPane();
    contentor.setBackground(Color.white);
}
public static void main(String[] args) {
  Janela1 j = new Janela1();
  j.setVisible(true);
 }
}// fim da classe Janela1
```


Subclasse de JFrame com fundo branco

Painel de conteúdos "content pane"

Designa a área da janela em que se pode mostrar conteúdo como texto, imagem, etc.

(i. é, toda a área, com exceção do título, da barra de menus e do bordo da janela)

Para aceder ao painel de conteúdos de uma janela usamos o método:

Container getContentPane(), definido na classe JFrame

O método devolve um objeto do tipo Container, classe do package java.awt

Adicionar botões, objetos do tipo JButton, ao painel de conteúdos de uma janela

Há duas formas de adicionar objetos gráficos no painel de conteúdos de uma janela:

1 – usar um objeto do tipo **LayoutManager** para controlar a colocação dos objetos

2 – explicitar o tamanho e a posição específica onde queremos o objeto (**posicionamento absoluto**)

Para usarmos a segunda possibilidade devemos desativar o layout manager, da classe Container, invocando o o método:

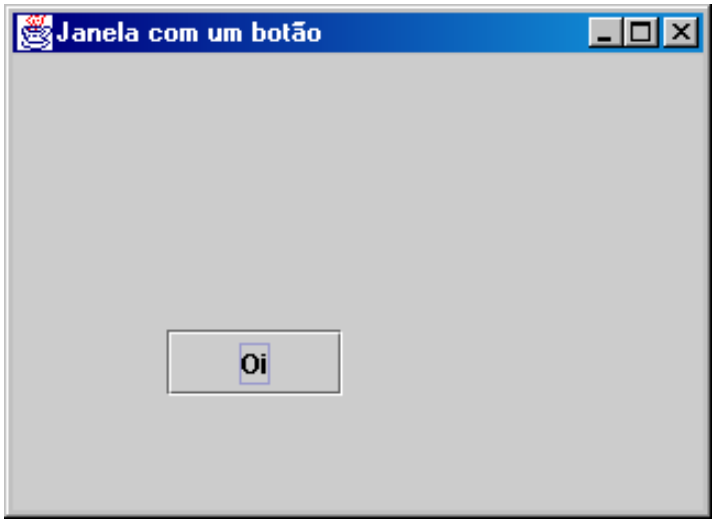
void setLayout(LayoutManager mgr) com o valor null

contentor.setLayout(null);

Para colocar o botão numa dada posição usamos o método da classe JButton:

void setBounds (x, y, width , height);

```
Container contentor = getContentPane();

JButton botao = new JButton(“Oi”);

contentor.setLayout(null);

botao.setBounds ( 70, 125, 80, 30 );

//Finalmente, vamos colocar o botão na janela:

contentor.add( botao );
```



Vamos agora ver como fazer o programa reagir
a um “click” sobre o botão ...

Programação por eventos

Um evento ocorre quando o utilizador interage com um objeto gráfico:

. manipular um botão com o rato;

. introduzir texto num campo de texto

. selecionar um item de menu

. ...

Num modelo de programação por eventos, programam-se objetos ("event listeners") que reagem a alterações no estado de outros objetos ("event sources"). Um "event listener" inclui um método que será executado em resposta ao (aos) evento(s) gerado(s).

Para tratar um evento é necessário associar (registar) um ou vários “listeners” ao objeto que gera o evento.

Quando um evento é gerado o sistema de execução notifica o objeto de escuta (“listener”) correspondente, invocando o método que trata o evento.

Caso não exista nenhum “listener” registado no objecto que gera o evento, este não terá qualquer efeito.

Para cada tipo de evento temos um “listener” correspondente.

Um objeto pode ser registado como um “listener” se for uma instância de uma classe que implementa uma determinada interface.

interface ActionListener

interface MouseListener

interface KeyListener

...

Exemplo:

```
    public interface ActionListener {
            public void actionPerformed(ActionEvent e);
    }

   import java.awt.event.*;
    public classe TrataBotao implements ActionListener {
        public void actionPerformed ( ActionEvent e ){
                    // ... Tratar evento
        }
}
```

Para registar um objeto "ActionListener" no objeto que gera o evento, usamos o método:

void addActionListener (ActionListener)

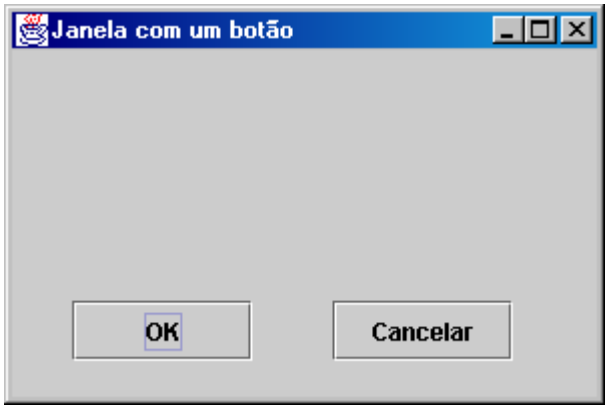
```
b1 = new JButton ( "OK“ );

TrataBotao  tb= new TrataBotao( );

b1.addActionListener ( tb );
```

Obs.

Um único "listener" pode ser associado a múltiplos objetos geradores de eventos.

Vários "listeners" podem ser associados a um único evento

Suponhamos a seguinte janela com dois botões:



Tratar eventos

Queremos que o título da janela mude consoante o botão que for pressionado. O método que vai tratar o evento terá a seguinte estrutura:

```
public void actionPerformed ( ActionEvent e ){

    (1) String textoDoBotao =
          aceder ao texto do objeto que gerou o evento;

    (2) JFrame janela =
          aceder à janela que contém o objeto gerador do evento
          janela.setTitle ("Pressionou o botão " +  textoDoBotao );
}
```

(1) Pode obter-se de duas formas:

a) pelo método "String getActionCommand()", da classe ActionEvent:

```
String textoDoBotao = e.getActionCommand ();
```

b) através do método "Object getSource()", da classe ActionEvent:

```
JButton botaoClicado = (JButton) e.getSource();
String textoDoBotao = botaoClicado.getText();
```

(2) Para obtermos a janela que contém o gerador do evento, é necessário:

. Obter o painel que contém o objeto gerador do evento

. Obter a janela que contém o painel.

```
JRootPane rootPane =  botaoClicado.getRootPane();

JFrame frame = (JFrame)  rootPane.getParent();
```

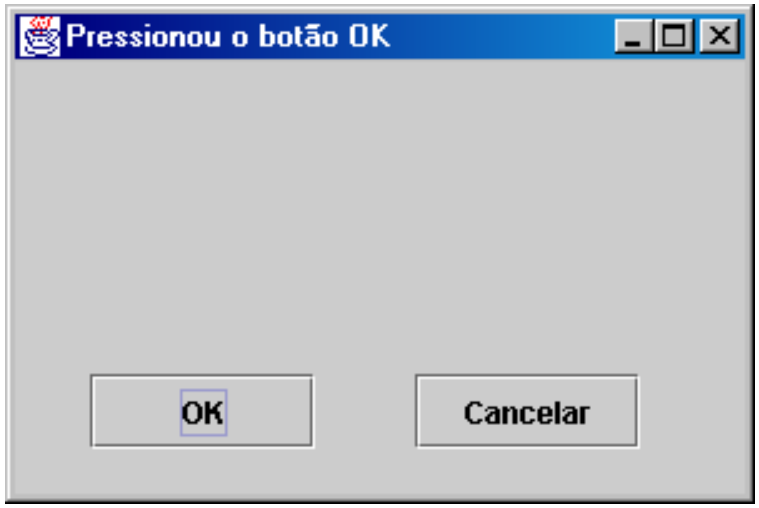
Uma janela pode conter vários painéis encadeados. O painel de topo é o “root pane”

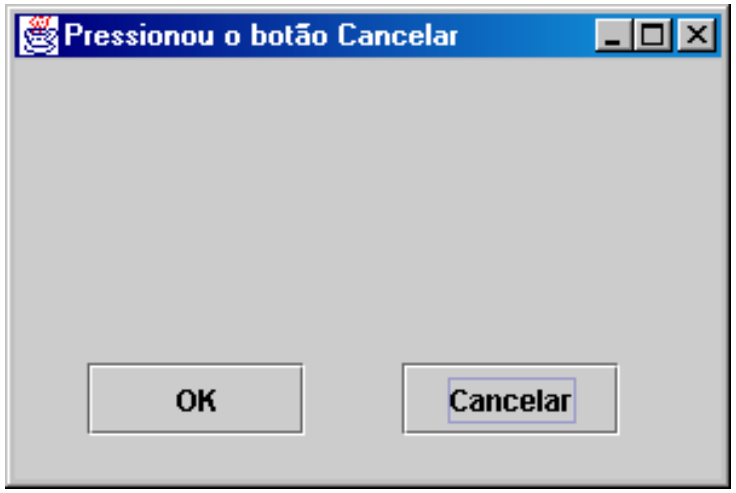
A própria janela pode ser o “listener” dos objetos gráficos que contém.

```
public class Janela2 extends JFrame implements ActionListener
    {
  private JButton b1, b2;
  ...
public Janela2(){
     ...
    Container contentor = getContentPane();
    b1 = new JButton("OK");
    b2 = new JButton("Cancelar");
```

```
//registar o listener em cada botão
   b1.addActionListener(this);
   b2.addActionListener(this);

   contentor.add(b1);
   contentor.add(b2);
  }

//Continua …
```

```
public void  actionPerformed(ActionEvent e){
    String textoDoBotao;
    textoDoBotao = e.getActionCommand();
    setTitle("Pressionou o botão " + textoDoBotao);
 }

public static void main(String[] args)  {
  Janela2 j = new Janela2();
  j.setVisible(true);
 }
} // fim da classe Janela2
```


Pressionou o botão OK
OK
Cancelar


Pressionou o botão Cancelar
OK
Cancelar

A classe JLabel

. Permite mostrar texto não editável

```
public class Janela3 extends JFrame {
   private JLabel l;
   ...
 public Janela3(){
    ...
 l = new JLabel("Vingança !!!!");
 l.setBounds(205,100,70,20);
contentor.add(l);
```

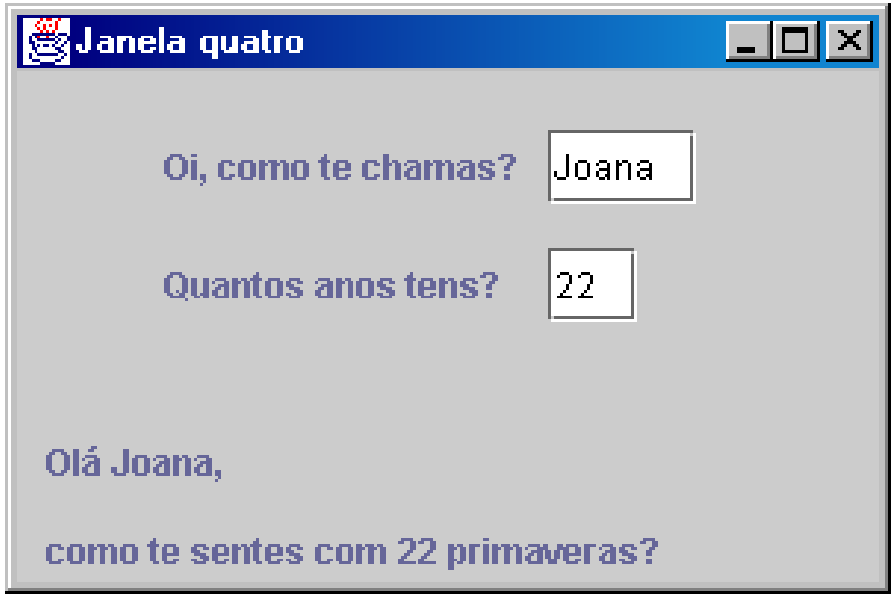
```
li = new JLabel (new ImageIcon ("d://_POO/rato.jpg"));

li.setBounds(0,0,200,200);

contentor.add(li);
```

A classe JTextField

. Um campo de texto, permite ao utilizador introduzir uma única linha de texto

. Uma instância da classe JTextField gera uma instância da classe ActionEvent quando o objeto está ativo e o utilizador pressiona a tecla ENTER.

Suponhamos que queremos construir a janela seguinte:



Tratando os eventos gerados quando o utilizador pressiona a tecla ENTER em cada um dos campos de texto.

```
public class Janela4 extends JFrame implements ActionListener{
 ...
private JLabel lNome, lIdade, lNome2, lIdade2;
private JTextField nome, idade;
private Container contentor;

 public Janela4(){
 ...
contentor = getContentPane();
 contentor.setLayout(null);
```

```
// criar a etiqueta com a 1º pergunta
lNome = new JLabel("Oi, como te chamas? ");
lNome.setBounds(50,20,125,25);
contentor.add(lNome);
// criar o campo de texto para o nome
nome = new JTextField();
nome.setBounds(180,20, 50,25 );
contentor.add(nome);
// registar o “listener” no campo de texto
nome.addActionListener(this);
// repetir para a 2ª pergunta (atenção às coordenadas)
... }
```

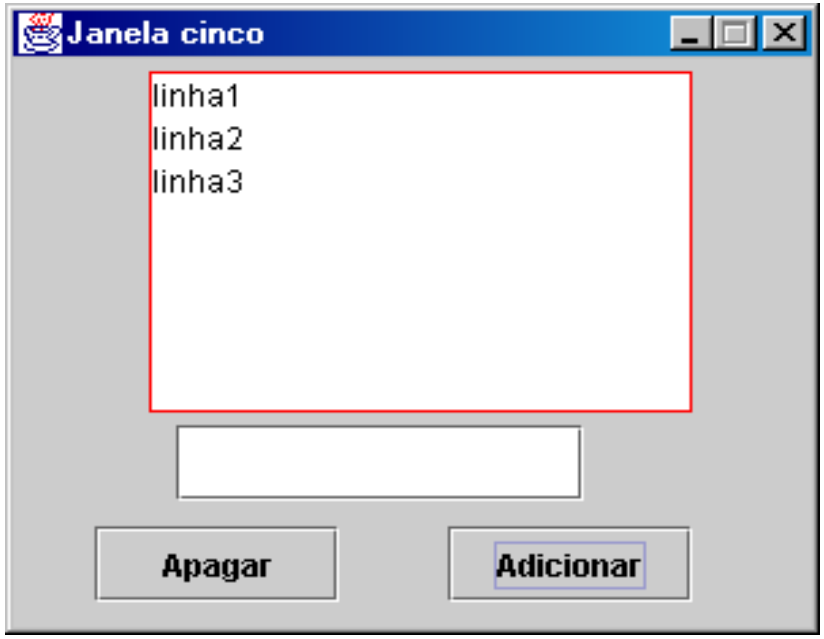
```
public void  actionPerformed(ActionEvent evt){

   JTextField campoTexto = (JTextField) evt.getSource();

   int w = campoTexto.getWidth();

   String texto = campoTexto.getText();
   if (w==50)
     mostraNome(texto);
    else
     mostraIdade(texto);
 }
```

```
public void mostraNome(String n){
 lNome2 = new JLabel("Olá " + n +",");
 lNome2.setBounds(10,120,100,25);
 contentor.add(lNome2);
}
public void mostraIdade(String n){
 int i = Integer.parseInt(n);
 lIdade2 =
      new JLabel("como te sentes com "+ i + “primaveras?");
 lIdade2.setBounds(10,150,250,25);
 contentor.add(lIdade2);
}
```

A classe JTextArea

. Uma área de texto permite introduzir várias linhas de texto

Suponhamos a seguinte janela:



Cada linha de texto introduzida no campo de texto será adicionada à área de texto sempre que, o utilizador

pressionar a tecla RETURN ou

o botão Adicionar.

Quando pressionado o botão Apagar será eliminado todo o conteúdo da área de texto.

```
private JButton botaoApagar;
private JButton botaoAdicionar;
private JTextField linha;
private JTextArea texto;


// no construtor criar a área de texto
 texto = new JTextArea();
 texto.setBounds(50,5,200, 135);
```

```
// por omissão a área de texto não é delimitada
 texto.setBorder(BorderFactory.createLineBorder (Color.red));
 texto.setEditable(false);
 contentor.add(texto);
```

Para adicionar texto à área de texto usamos o método
void append (String) da classe JTextArea

Para substituir o texto da área de texto usamos o método
void setText (String) da classe JTextArea

```
public void  actionPerformed(ActionEvent evt){
   if ( evt.getSource() instanceof JButton ) {
     JButton  botao = (JButton) evt.getSource();
     if ( botao == botaoAdicionar ) {
       adicionarTexto ( linha.getText() ) ;
     }else {
       apagarTexto();
     }
   } else {// event source: linha
     adicionarTexto(linha.getText() );
   }
 }
```

```
private void adicionarTexto(String s){
    texto.append( s + "\n");
    linha.setText("");
 }

 private void apagarTexto(){
    texto.setText("");
    linha.setText("");
}
```

Scroll bars

Para permitir a visualização de um texto que exceda as margens,

modificamos o código anterior onde criámos a área de texto englobando esta num objeto do tipo "JScrollPane":

```
texto = new JTextArea();
 texto.setEditable(false);

JScrollPane elevador = new JScrollPane (texto);
 elevador.setBounds (50,5,200, 135);
 elevador.setBorder (
            BorderFactory.createLineBorder (Color.red));
 contentor.add(elevador);
```

As classes JMenuBar, JMenu, JMenuItem

. Uma barra de menus (JMenuBar) é a barra onde são colocados os menus

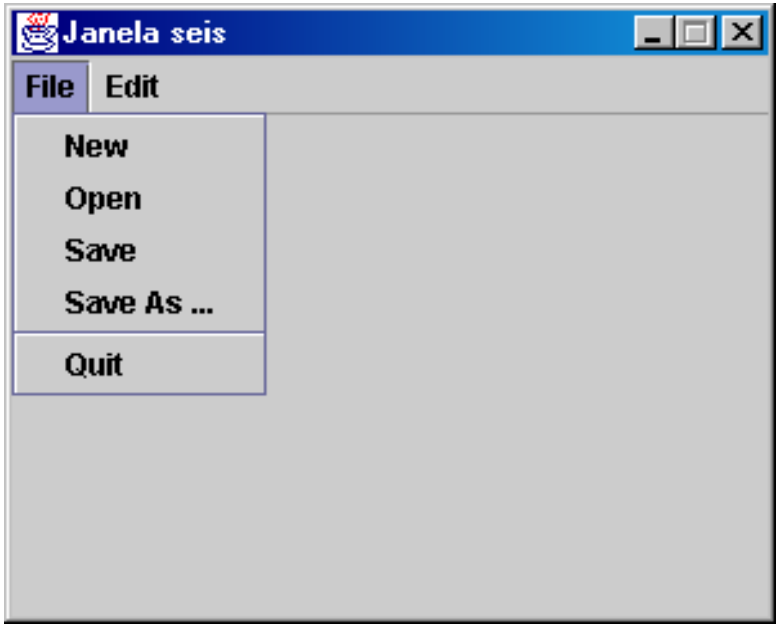
. Um menu (JMenu) é um item de uma barra de menus

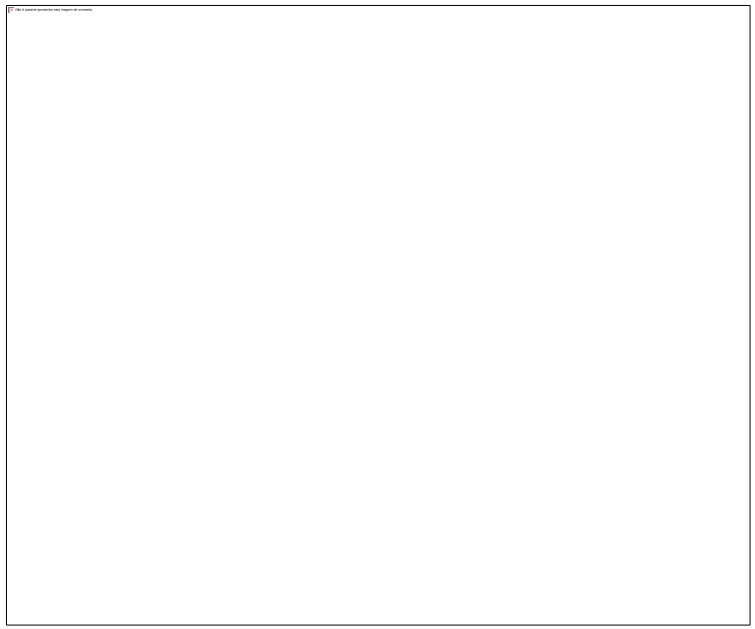
. Um item de menu (JMenuItem) é uma das opções de um menu

Quando um item de um menu é selecionado, é gerado um "action event"

Para processar a seleção de um item de um menu associamos um "listener" a esse item.

Vamos construir uma janela com dois menus:





Sequência de passos para criar os menus:

1 – criar uma barra de menus e adicionar à janela

2 – criar os menus

3 – criar os itens de menu e adicionar ao menu correspondente

4 – adicionar cada menu à barra de menus

```
public class Janela6 extends JFrame implements ActionListener{
...
private JLabel resposta;
private JMenu fileMenu, editMenu;
public Janela6(){
...
```

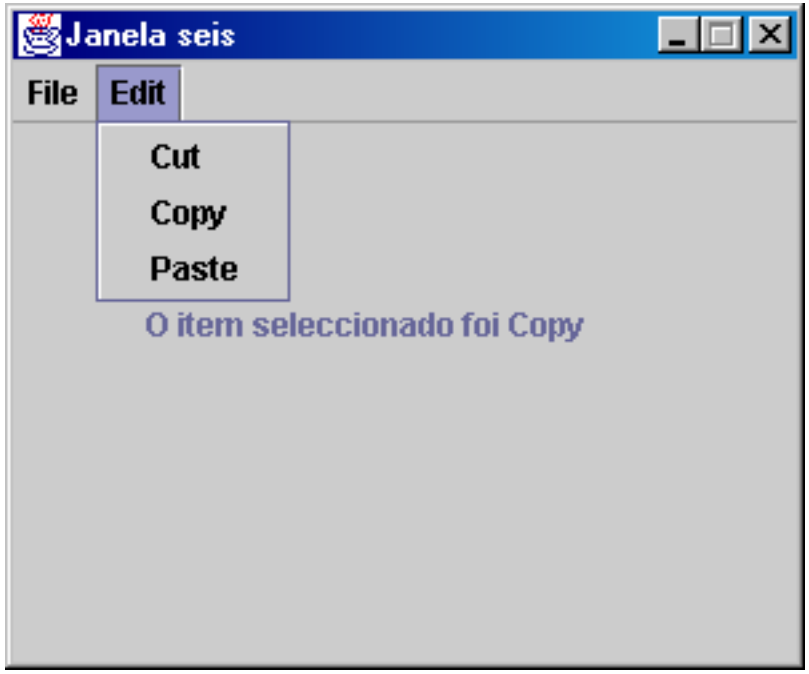
```
// criar a barra de menus
 JMenuBar barraMenus = new JMenuBar();
 //criar dois menus e os seus items
 criarFileMenu();
 criarEditMenu();

 //colocar a barra de menus na janela
 setJMenuBar ( barraMenus);

 // adicionar os menus à barra de menus
 barraMenus.add(fileMenu);
 barraMenus.add(editMenu);
```

```
public void criarFileMenu(){
JMenuItem item;
 fileMenu= new JMenu("File");

 item = new JMenuItem( "New");
 item.addActionListener(this);
 fileMenu.add(item);

 item = new JMenuItem( "Open");
 item.addActionListener(this);
 fileMenu.add(item);
```

```
 item = new JMenuItem( "Save");
 item.addActionListener(this);
 fileMenu.add(item);
 ...
 fileMenu.addSeparator(); // linha horizontal

 item = new JMenuItem( "Quit");
 item.addActionListener(this);
 fileMenu.add(item);
}

// análogo para o método criarEditMenu
```

Tratar a selecção de um item

```
public void  actionPerformed(ActionEvent evt){
   String itemSeleccionado;
   itemSeleccionado = evt.getActionCommand();
   if ( itemSeleccionado.equals("Quit") ) {
      System.exit(0);
   } else {
      resposta.setText("O item seleccionado foi " +
      itemSeleccionado );
   }
 }
```


Janela seis
File
Edit
Cut
Copy
Paste
O item seleccionado foi Copy