

## PG II

# Programação Orientada aos Objectos em Java

### **AWT - Abstract Window Toolkit:**

- Introdução
- Componentes AWT
- *Containers e Layout Managers*
- Modelo de Eventos

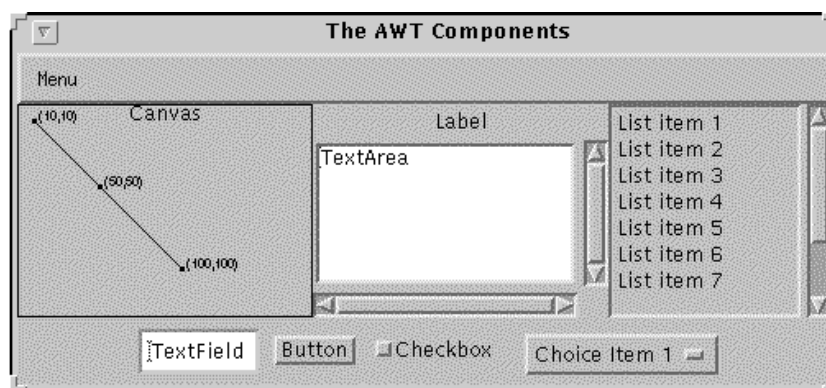
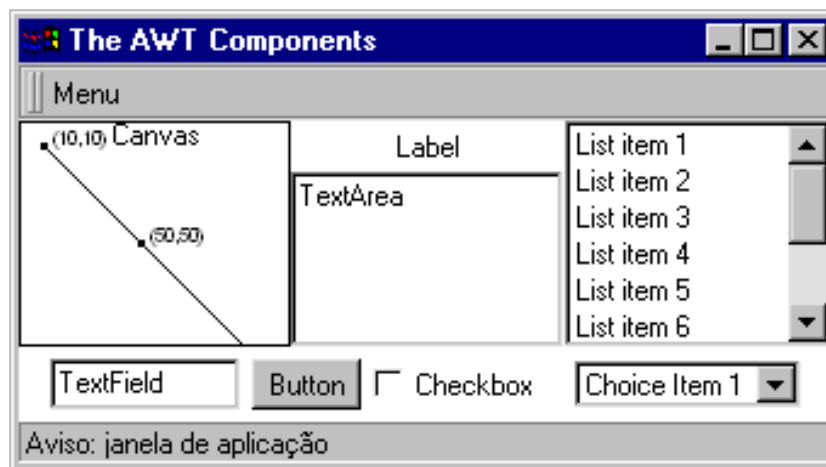
Os slides seguintes apresentam alguns conteúdos do documento: “Interface gráfica em Java”, do Eng. Pedro Pereira.

- Consiste numa *framework* para desenvolvimento **GUI - Graphical User Interface**;
- Faz parte das **JFC - Java Foundation Classes** – que são um conjunto de bibliotecas de suporte ao desenvolvimento **GUI** em aplicações Cliente (*based client*);
- Além do **AWT** fazem parte das **JFC**:
  - **Java 2D** - Desenvolvimento avançado de gráficos 2D, imagens, texto e impressões.
  - **Swing GUI Components** – Extensão ao AWT, que disponibiliza componentes mais “enriquecidos” e com um novo “*look anda feel*”.
  - **Accessibility API** - Tecnologias assistidas de interacção e comunicação entre componentes JFC,
  - **Internationalization** - Suporte às convenções culturais de cada país.

# Componentes AWT

A interface gráfica AWT consiste num conjunto de componentes (*widgets*), representados por classes. Os principais componentes são:

- **Button**
- **Canvas**
- **Checkbox**
- **Choice**
- **Label**
- **List**
- **Scrollbar**
- **TextArea**
- **TextField**

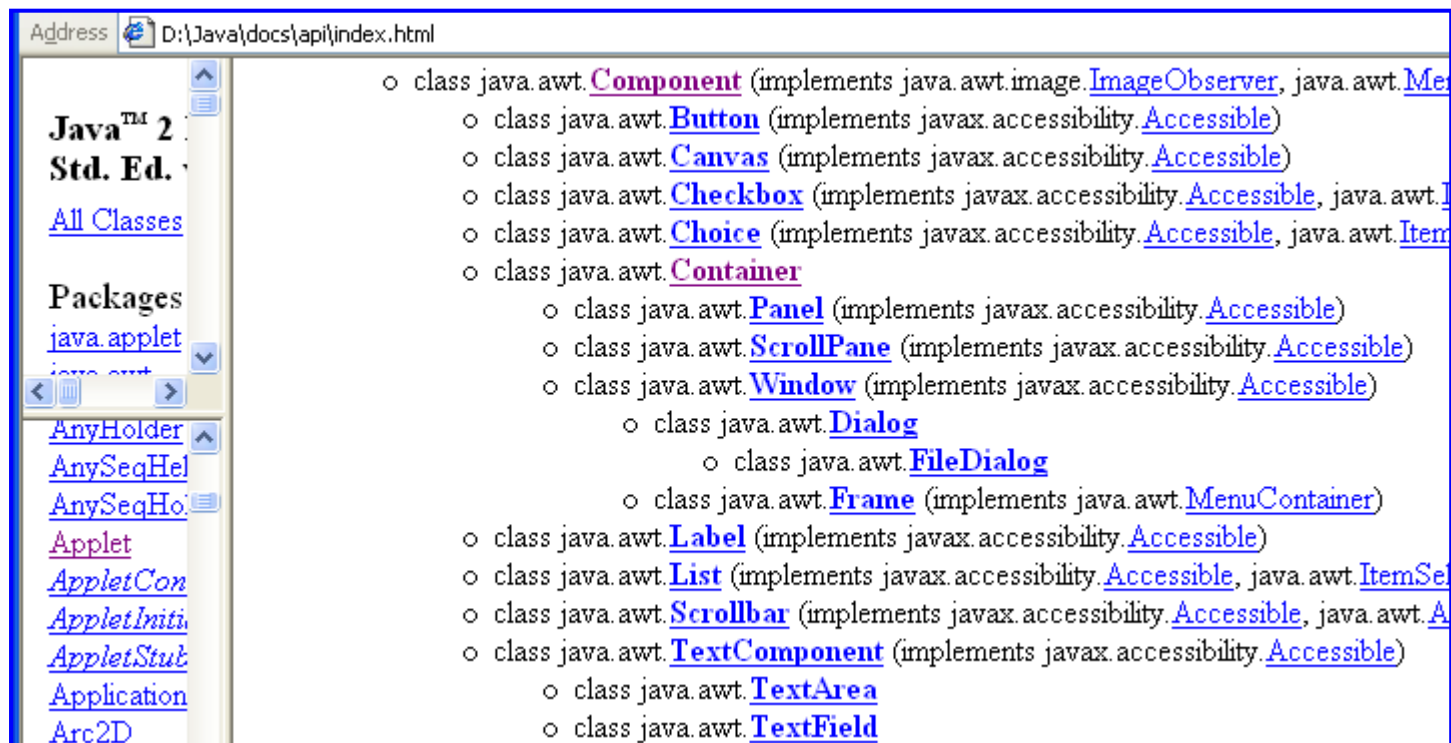


**A mesma  
aplicação  
a executar  
no  
Windows  
e  
no  
Solaris**

## ... Componentes AWT

No “**Java API Specification**” os componentes AWT aparecem organizados da seguinte forma:

- **Button**
- **Canvas**
- **Checkbox**
- **Choice**
- **Label**
- **List**
- **Scrollbar**
- **TextArea**
- **TextField**



## ... Componentes AWT

Objecto com uma representação gráfica que pode ser apresentado no ecrã e interactivar com o utilizador.

- Posição e Área
  - [set][get]Size, get[Maximum][Minimum][Preferred]Size, [set][get]Location, [set][get]X, [set][get]Y, getWidth, getHeight, contains, [set][get]Bounds, getParent
- Aspecto
  - [set][get]Background, [set][get]Foreground, isOpaque, [set][get]Font, [set][get]Cursor, update, paint, repaint, print
- Reacção a estímulos
  - disableEvents, enableEvents, process???Event
  - add???Listener, remove???Listener
- Estado
  - [set][is]Enabled, [set][is]Visible, isShowing

### Classes Auxiliares:

- **Rectangle, Dimension, Point, Image, Graphics, Font e Color.**

# Containers e Layout Managers

Os componentes por si só não são suficientes para a construção de janelas. *Container* é um componente que pode conter componentes.

Existem dois tipos de *Containers*: **Panel** e **Window**.

```
○ class java.awt.Component (implements java.  
    ○ class java.awt.Container  
        ○ class java.awt.Panel (implemen  
            ○ class java.applet.Applet
```

- **Panel**
  - Rectângulo que contem componentes.
  - Applet é uma classe derivada desta.

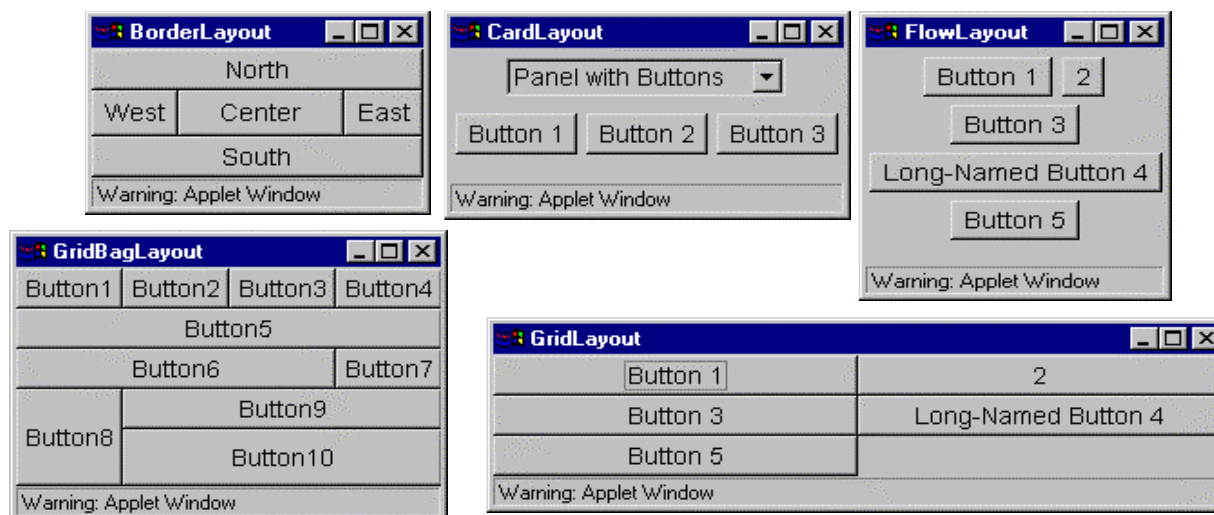
```
○ class java.awt.Container  
    ○ class java.awt.Panel (implements javax.  
    ○ class java.awt.ScrollPane (implements j  
    ○ class java.awt.Window (implements java  
        ○ class java.awt.Dialog  
            ○ class java.awt.FileDialog  
        ○ class java.awt.Frame (implement
```

- **Window**
  - Janela com o aspecto das janela da plataforma de execução.
  - De onde derivam classes como *Frame*, *Dialog* e *FileDialog*.

## ... Containers e Layout Managers

Os componentes adicionados a um determinado **Container** são organizados de acordo com a especificação de um determinado **LayoutManager**:

- BorderLayout;
- CardLayout;
- FlowLayout
- GridBagLayout;
- GridLayout.



- Qualquer `Container` tem um *layout* por omissão:
  - **Panel:** `FlowLayout`
  - **Window:** `BorderLayout`
- O método `add(Component)` invoca indirectamente o *layout* usado.
- O método `setLayout(LayoutManager mgr)`, modifica o layout manager do contentor para o parâmetro especificado.



## ... Containers e Layout Managers

```

Button2

package pg2.aula06;
import java.awt.*;

public class Button2 extends Frame {

    //Instance Variables
    Button b1 = new Button("First");
    Button b2 = new Button("Second");
    Label l = new Label("No button pressed",Label.CENTER);

    public Button2() {
        super("Button 2");

        // <<Composição>>
        Panel buttons = new Panel();
        buttons.add(b1); buttons.add(b2);
        add(l); add(buttons,BorderLayout.SOUTH);
    }
}

```

```

Button2 Application

package pg2.aula06;

public class Application {

    public static void main (String [] args){
        Button2 f = new Button2();
        f.pack();
        f.setVisible(true);
    }
}

```



## ... Containers e Layout Managers

```

AppletButton

package pg2.aula06;

import java.awt.*;
import java.applet.*;

public class AppletButton extends Applet {

    //Instance Variables
    Button b1 = new Button("First");
    Button b2 = new Button("Second");
    Label l = new Label("No button pressed",Label.CENTER);

    /**Construct the applet*/
    public AppletButton() {
        this.setLayout(new BorderLayout());
        Panel buttons = new Panel();
        buttons.add(b1); buttons.add(b2);
        add(l);
        add(buttons,BorderLayout.SOUTH);
    }
}

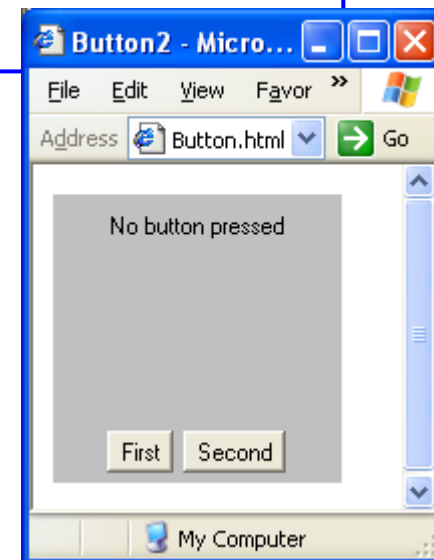
```

```

AppletButton.html

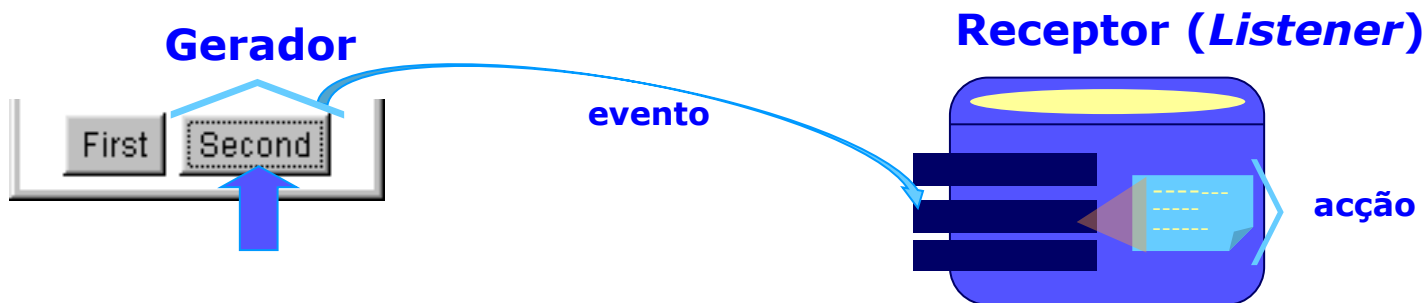
<HTML>
<HEAD> <TITLE>
    Button2
</TITLE> </HEAD>
<BODY>
    <APPLET
        CODE      = "pg2.aula06.AppletButton.class"
    </APPLET>
</BODY>
</HTML>

```



# Modelo de Eventos

Uma das tarefas na implementação de um GUI (*Graphical User Interface*), é o desenvolvimento das **acções desencadeadas** por cada um dos componentes desse interface.



- No modelo de objectos AWT, as **acções** são desencadeadas por **eventos**.
- Neste modelo existem três intervenientes principais:
  - O evento;
  - O gerador do evento (**Source**);
  - O receptor do evento (**Listener**).

## ... Modelo de Eventos



O mecanismo de associação entre o **Gerador do evento** e o respectivo **Listener** procede-se da seguinte forma:

**Gerador**      **Evento**      **Listener**

```
b2.addActionListener (new Handler())
```

- O **Gerador do Evento**, neste exemplo o botão com a label "Segundo" (referenciado pela variável **b2**), irá indicar qual o **Listener** que ficará como "*handler*" do evento gerado;
- **Action – evento** desencadeado (objecto da classe **ActionEvent** que deriva de **AWTEvent**);
- **new Handler** – (Listener) instância da classe Handler, que implementa obrigatoriamente a Interface **ActionListener** (caso contrário é gerado um erro de compilação).  
Nesta situação em que o Listener e o Gerador do evento fazem parte da mesma classe, faz sentido declarar o Listener como Inner Class da classe geradora do evento.

## ... Modelo de Eventos

```

AppletButton Button2
package pg2.aula06;

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

public class Button2 extends Frame {

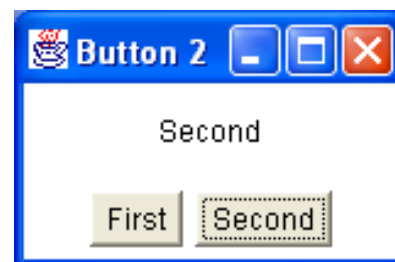
    //Instance Variables
    Button b1 = new Button("First");
    Button b2 = new Button("Second");
    Label l = new Label("No button pressed",Label.CENTER);

    public Button2() {
        super("Button 2");

        // <<Composição>>
        Panel buttons = new Panel();
        buttons.add(b1); buttons.add(b2);
        add(l); add(buttons,BorderLayout.SOUTH);

        // <<Comportamento>>
        addWindowListener( new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); }
        });
        ActionListener al = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                l.setText( e.getActionCommand() );
            }
        };
        b1.addActionListener( al );
        b2.addActionListener( al );
    }
}

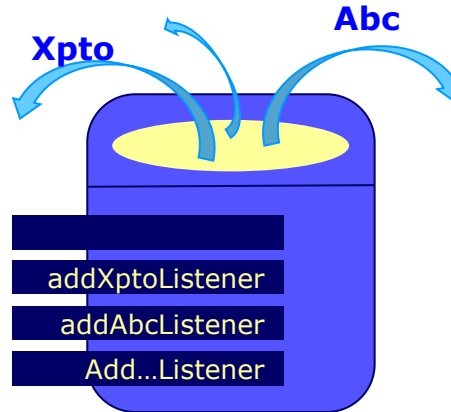
```



## ... Modelo de Eventos - Gerador



Os diversos componentes AWT são geradores de eventos. Todos os tipos de **eventos XXX**, desencadeados por um determinado **componente** terão um método correspondente **addXXXListener**.



Qualquer componente gera eventos do tipo **Component**, **Focus**, **Key** e **Mouse** :

Objectos desta classe ou derivadas	Geram eventos do Tipo
Button	Action
List	Action, Item
Window	Window
Choice, Checkbox	Item
Container	Container
TextComponent	Text

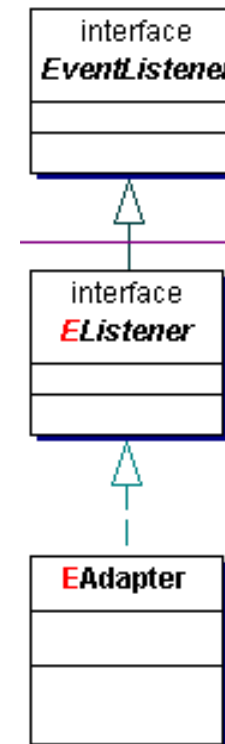
## ... Modelo de Eventos - Listener



O **receptor do evento**, vulgarmente designado como "**Listener**", trata-se de um objecto de uma classe que implemente a interface **XXXListener** ou derive da classe **XXXAdapter**.

### Interface Hierarchy

- interface java.util. [EventListener](#)
- interface java.awt.event. [ActionListener](#)
- interface java.awt.event. [AdjustmentListener](#)
- interface java.awt.event. [AWTEventListener](#)
- interface java.awt.event. [ComponentListener](#)
- interface java.awt.event. [ContainerListener](#)
- interface java.awt.event. [FocusListener](#)
- interface java.awt.event. [HierarchyBoundsListener](#)
- interface java.awt.event. [HierarchyListener](#)
- interface java.awt.event. [InputMethodListener](#)
- interface java.awt.event. [ItemListener](#)
- interface java.awt.event. [KeyListener](#)
- interface java.awt.event. [MouseListener](#)
- interface java.awt.event. [MouseMotionListener](#)
- interface java.awt.event. [TextListener](#)
- interface java.awt.event. [WindowListener](#)



Cada um destes eventos tem declarados os métodos, correspondentes às mensagens enviadas, para cada um dos **evento gerados**.

## ... Modelo de Eventos - Evento



O **evento** é um objecto de uma Classe, que deriva da classe **AWTEvent**.

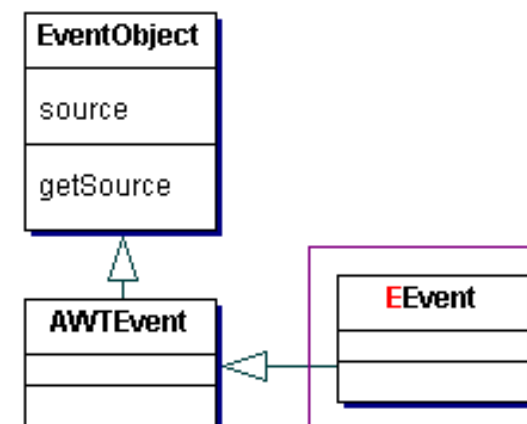
### Hierarchy For Package java.awt.event

#### Package Hierarchies:

[All Packages](#)

#### Class Hierarchy

- class java.lang. **Object**
  - class java.awt.event. **ComponentAdapter** (implements java.awt.event. **ComponentListener**)
  - class java.awt.event. **ContainerAdapter** (implements java.awt.event. **ContainerListener**)
  - class java.util. **EventObject** (implements java.io. **Serializable**)
    - class java.awt. **AWTEvent**
      - class java.awt.event. **ActionEvent**
      - class java.awt.event. **AdjustmentEvent**
      - class java.awt.event. **ComponentEvent**
        - class java.awt.event. **ContainerEvent**
        - class java.awt.event. **FocusEvent**
        - class java.awt.event. **InputEvent**
          - class java.awt.event. **KeyEvent**
          - class java.awt.event. **MouseEvent**
        - class java.awt.event. **PaintEvent**
        - class java.awt.event. **WindowEvent**

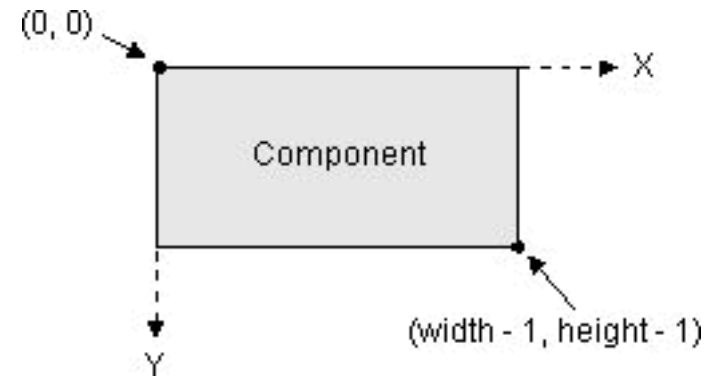




# O contexto gráfico – Classe Graphics

O objecto da classe *Graphics* passado como parâmetro a `paint()` e `update()` tem a informação necessária para as operações gráficas suportadas no Java:

- O Componente AWT onde se está a desenhar.
- A translação da origem das coordenadas.
  - (0,0) é o canto superior esquerdo.
- A área de *clipping* corrente.
- A cor corrente.
- A fonte corrente.



- A classe *Graphics* tem um conjunto de métodos para desenhar figuras (ocas ou a cheio).



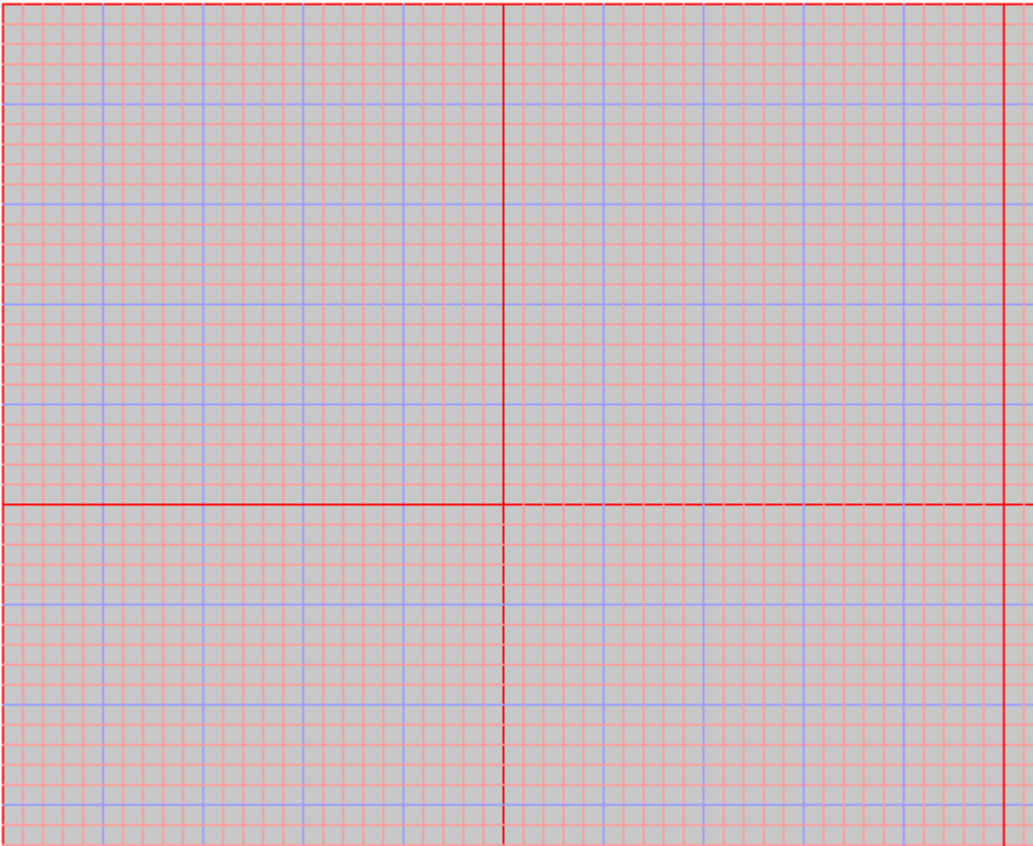
Na maioria dos métodos é passado como parâmetro as coordenadas do rectângulo em que a figura fica inserida.

# Exemplo do 2º Trabalho

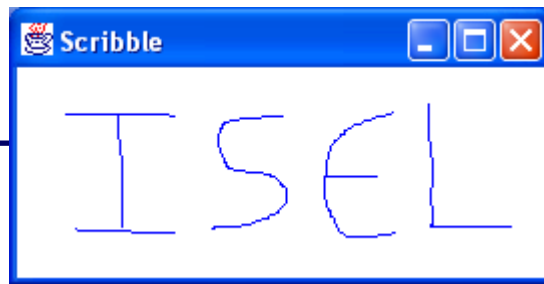


Área e perímetro de um polígono

Ponto  (X =  , Y =  )



0.00000000  0.00000000



```
public class Scribble extends Canvas {
    static private class Line {
        public Point s, d;
        public Line( Point a, Point b ) {
            s = a; d = b;
        }
    }
    private List lines = new LinkedList();
    private class MouseManager ...
    public Scribble() {
        MouseManager mm = new MouseManager();
        addMouseListener( mm );
        addMouseMotionListener( mm );
        setForeground( Color.blue );
    }
    public void paint( Graphics g ) {
        for(Iterator e=lines.iterator();e.hasNext();) {
            Line l = (Line) e.next();
            g.drawLine( l.s.x, l.s.y, l.d.x, l.d.y );
        }
    }
    public static void main ( String[] args ) {
        ...
    }
}
```

```
private class MouseManager extends MouseAdapter
    implements MouseMotionListener {
    public void mouseMoved(MouseEvent e) {}
    private Point last;
    public void mousePressed(MouseEvent e) {
        last = e.getPoint();
    }
    public void mouseDragged(MouseEvent e) {
        Point p = e.getPoint();
        lines.add( new Line( last, p ) );
        Graphics g = getGraphics();
        g.drawLine( last.x , last.y , p.x, p.y );
        last = p;
    }
}
```

```
public static void main( String[] args ) {
    Frame f = new Frame("Scribble");
    f.add( new Scribble() );
    f.addWindowListener( new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    } );
    f.pack();    f.setVisible(true);
}
```

- class java.awt.[MenuComponent](#) (implements java.io.[Serializable](#))
  - class java.awt.[MenuBar](#) (implements javax.accessibility.[Accessible](#), java.awt.[MenuContainer](#))
  - class java.awt.[MenuItem](#) (implements javax.accessibility.[Accessible](#))
    - class java.awt.[CheckboxMenuItem](#) (implements javax.accessibility.[Accessible](#), java.awt.[ItemSelectable](#))
    - class java.awt.[Menu](#) (implements javax.accessibility.[Accessible](#), java.awt.[MenuContainer](#))
      - class java.awt.[PopupMenu](#)
- class java.awt.[MenuShortcut](#) (implements java.io.[Serializable](#))