UAST
Unidade Acadêmica de Serra Talhada - PE
UFRPE
Desde 2006

BSi
Bacharelado em Sistemas de Informação

MPOO

Site: https://sites.google.com/site/profricodemery/mpoo

Site: http://ava.ufrpe.br/

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery

LISTA DE EXERCÍCIOS VII

Leia atentamente as instruções gerais:

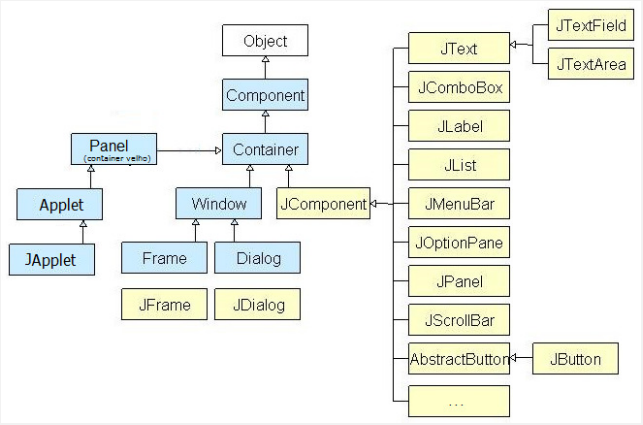
- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listaVII.SeuNomeSobrenome**, o qual deverá ter **pastas de pacotes** para cada questão: questao1, questao2, e assim sucessivamente, contendo todas as respostas da lista.
- A lista envolve questões práticas, então deverão ser entregues no AVA os códigos-fonte (projeto completo).
- A entrega da lista compõe sua frequência e avaliação na disciplina.

Fique atento!

Prezado aluno, nesta lista de exercícios exploraremos a construção de interfaces gráficas para usuários. Por isso, antes de responder, reveja a vídeo-aula sobre "Componentes Gráficos".

Saiba Mais!

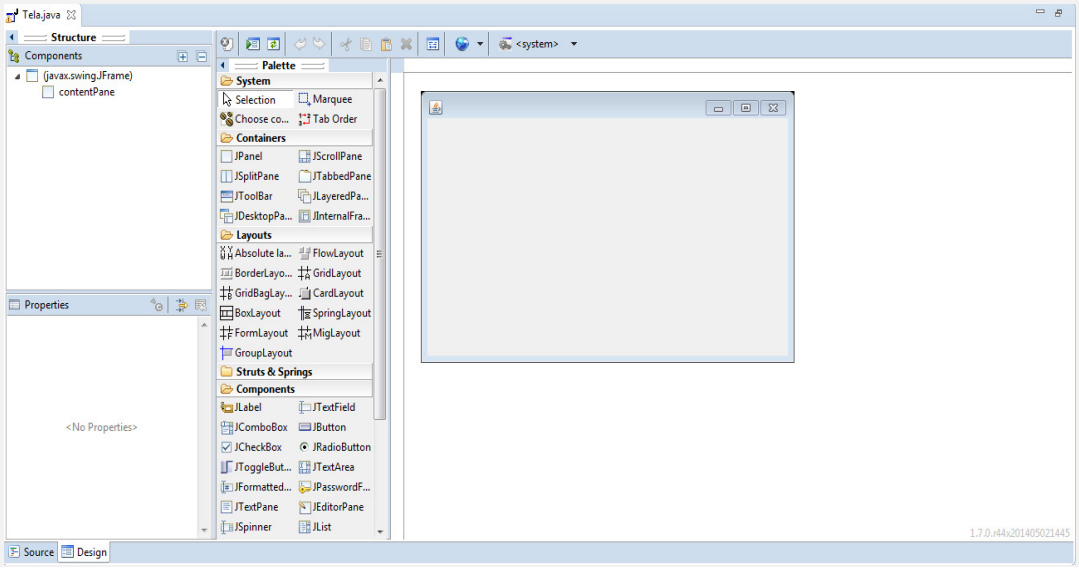
Os elementos básicos necessários para criar uma interface gráfica (GUI, do inglês *graphical user interface*) para um usuário utilizar um sistema estão em dois pacotes java.awt e javax.swing, em que awt foi o primeiro conjunto de classes Java para construir GUI's, enquanto swing é uma extensão de awt que mantem e amplia os conceitos de awt, em especial para tratar as aparências multiplataformas.



Uma GUI é baseada em dois elementos: containers (janelas e painéis) e componentes (menus, botões, caixas de texto e seleção, barras de rolagem, rótulos, tabelas, etc.).

Investigue e pratique a implementação de diversos componentes: https://docs.oracle.com/javase/8/docs/api/javax/swing/JComponent.html

Visando a produtividade de desenvolvimento de software, diversos IDE's dispõem de editores visuais baseados em *drag-and-drop* (arrastar e soltar) para construção de GUI's.



Você Sabia?

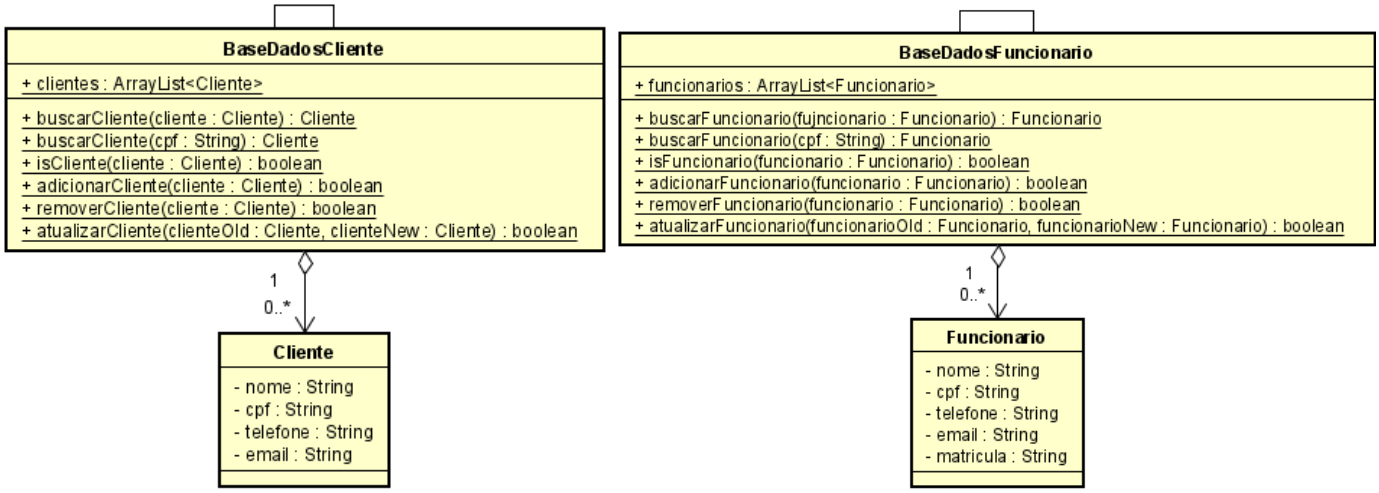
A aparência e a forma em que o usuário interage com a aplicação são chamados de look and feel da aplicação.

Desde a atualização 10 do Java SE 6, as GUI's passaram a ter uma *cara* nova, elegante e compatível com várias plataformas, conhecida como Nimbus.

```
try {
        UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
} catch (ClassNotFoundException | InstantiationException
          | IllegalAccessException | UnsupportedLookAndFeelException e) {
        e.printStackTrace();
}
```

Mão na Massa!

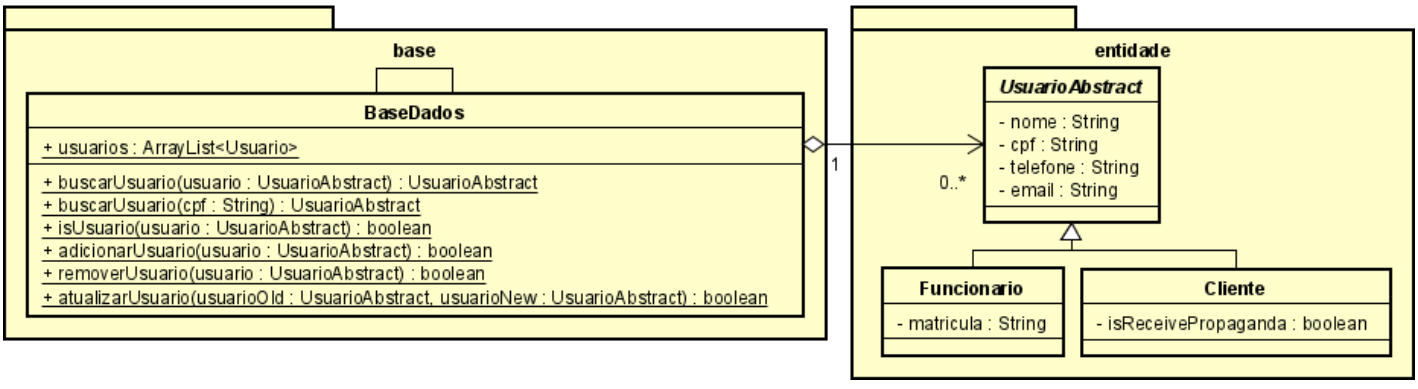
1) Vimos (em outra lista de exercícios) que o programador "O Furão" (codinome para *mustela putórius furo*) precisou resolver a demanda de um contratante da empresa MPOOSoftware LTDA para a atualização de um sistema de cadastro. Inicialmente se deparou com os seguintes diagramas de classes atuais da empresa:



Após analisar os diagramas e as regras de negócios:

- RN01 – um cliente ou funcionário é identificado pelo seu cpf;
- RN02 – um cliente ou funcionário só poderá ser cadastrado uma única vez; e
- RN03 – a empresa só envia propaganda se o cliente permitir recebê-la.

Propôs uma melhoria para o sistema (diagrama abaixo) que foi aceita por seu Scrum Master, fazendo com que passasse a ter uma única base com uso de polimorfismo.

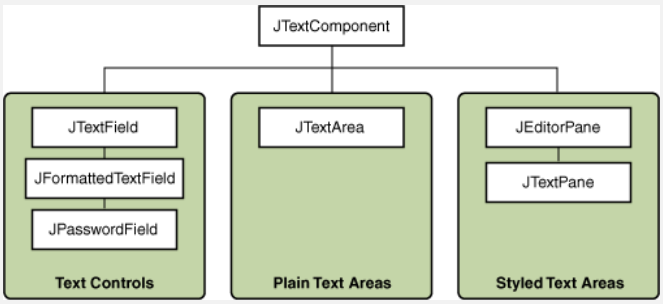


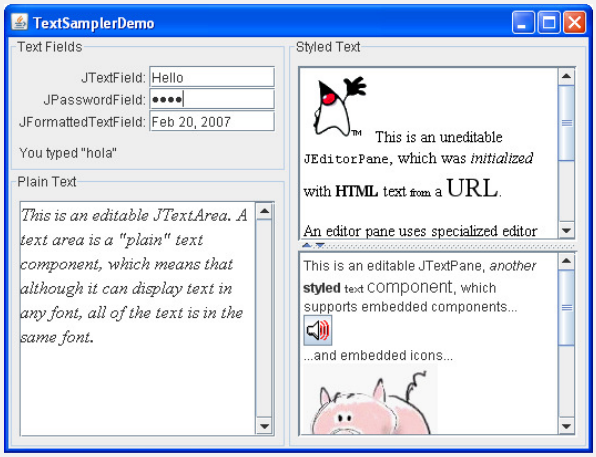
Sendo assim codifique a GUI ao lado **(Fig. 1A)** para que se possa cadastrar um usuário, sabendo que este pode ser um "Funcionario" ou "Cliente". Para o desenvolvimento, o Scrum Master definiu que a solução deve utilizar exclusivamente as bibliotecas disponíveis no JDK, ou seja, em `java` e `javax`, sem o auxílio de recursos de IDE's do tipo *drag-and-drop*, como, por exemplo, a *palette de design* do Eclipse.

Fig. 1A

Sabia Mais!

Os componentes de texto podem ser personalizados. O Swing fornece seis componentes de texto, através de classes e interfaces. Apesar de seus diferentes usos, todos herdam da mesma superclasse, JTextComponent, que fornece uma base altamente configurável e poderosa para manipulação de texto. Vide a hierarquia de JTextComponent e um exemplo de sua utilização.





Para saber mais sobre JTextComponent acesse o tutorial disponível em: https://docs.oracle.com/javase/tutorial/uiswing/components/text.html

Desafio!

São diversos os gerenciadores de layout que servem para determinar o tamanho e a posição dos componentes em uma GUI. Entretanto, são diversas as possibilidades, inclusive da utilização de soluções disponibilizadas em classes. Para desafiar "O Furão", seu Scrum Master o apresentou as possibilidades de configuração para JTextComponent, ao SpringLayout através de um exemplo (Fig. 1B) e de um tutorial de JavaSE da Oracle disponível em https://docs.oracle.com/javase/tutorial/uiswing/layout/spring.html. Mostre que você também é capaz de aplicar soluções de layouts, em especial, a tela de cadastro de usuário (Fig. 1A). Para isso, utilize a personalização pas os tipos de JTextComponent e a combinação de JPanels e gerenciadores de layout como BorderLayout e SpringLayout.

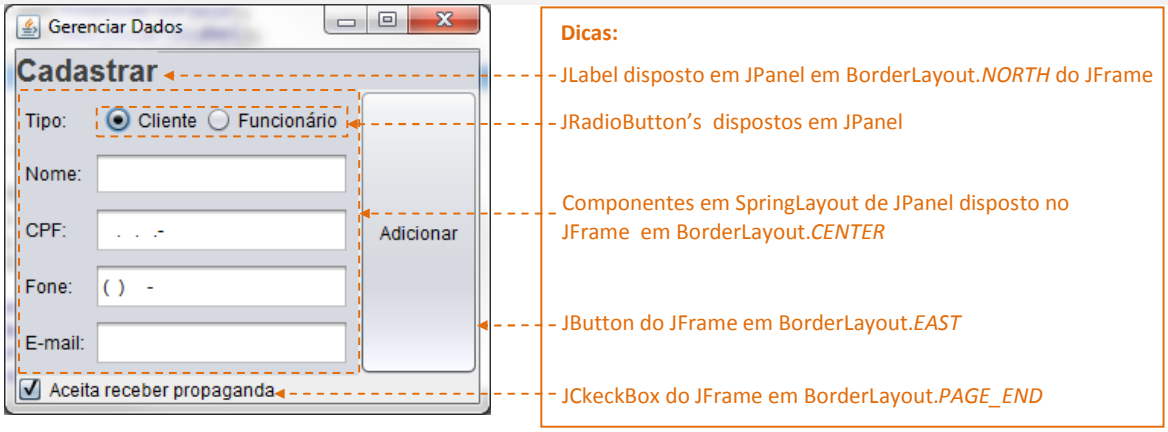


Fig. 1B

Fique Atento!

Observe a utilização do componente JRadioButton. Para que a seleção entre os tipos de usuários "Cliente" e "Funcioário" é de disjunção exclusiva (*exclusive or* - XOR). Com isso, os dados do cadastro depende se o usuário é um cliente ou um funcionário. Para que essa disjunção seja aplicada é necessário o agrupamento desses componentes em ButtonGroup.

Para que a seleção do tipo de usuário reflita no formulário, é necessário um Tratamento de Evento para as opções de JRadioButton.

Para saber mais sobre JRadioButton, ButtonGroup e seu Tratamento de Evento acesse a seção "How to Use Radio Buttons" do tutorial de JavaSE da Oracle disponível em https://docs.oracle.com/javase/tutorial/uiswing/components/button.html.

Mas antes de continuar com essa problemática vamos relembrar abaixo as possibilidades de tratar as interações de um usuário do sistema com a GUI.

Relembrando!

Na aula de MPOO introduzimos o assunto de componentes gráficos em que aprendemos a criar interfaces gráficas para os nossos sistemas. Também vimos diversas possibilidades de codificação para a interação com essas telas (Tratamento de Eventos!). Para ilustras as diferentes possibilidades, a tela de um sistema (GUI) será definida por View.

- Tratamento de evento em classe realizando uma interface

```
//View.java
public class View extends JFrame implements ActionListener {

    JButton button;

    public View(){
           //construção da GUI
           button = new JButton();
           //registro de listener
           button.addActionListener(this); //this -> indicação de tratamento na própria classe
    }

    //Método manipulador pertencente a classe
    @Override
    public void actionPerformed(ActionEvent event) {
           // método manipulador para tratar uma ação a ser executada por um componente gráfico, como,
por exemplo, button
    }
}
```

- Tratamento de evento por classe interna anônima

```
//View.java
public class View extends JFrame {

    JButton button;

    public View(){
           //construção da GUI
           button = new JButton();
           //registro de listener
           button.addActionListener(
                        new ActionListener() { //tratamento por classe interna anônima
                               @Override
                               public void actionPerformed(ActionEvent e) {
                                     // método manipulador para tratar a ação de button
                               }
                        });
    }
}
```

- Tratamento de evento por classe interna

```
//View.java
public class View extends JFrame{

    JButton button;
    ButtonHandler buttonHandler;

    public View(){
           //construção da GUI
           button = new JButton();
           buttonHandler = new ButtonHandler();//instância para classe interna
           //registro de listener
           button.addActionListener(buttonHandler); //indicação de tratamento na própria classe
    }

    private class ButtonHandler implements ActionListener{
           //Método manipulador pertence a classe interna
           @Override
           public void actionPerformed(ActionEvent arg0) {
                  // método manipulador para tratar uma ação a ser executada por um componente gráfico,
como, por exemplo, button
           }
    }
}
```

- Tratamento de evento por classe outra classe que não interna

```
//View.java
public class View extends JFrame{

    JButton button;
    ButtonHandler buttonHandler;

    public View(){
           //construção da gui
           button = new JButton();
           buttonHandler = new ButtonHandler(this); //passa esta view para a classe ButtonHandler que
tratará do evento ("comunicação entre classes!")

           //registro de listener
           button.addActionListener(buttonHandler); //indicação de tratamento na classe ButtonHandler
    }
}


//ButtonHandler.java
public class ButtonHandler implements ActionListener{

    View view;

    public ButtonHandler(View view) {
           this.view = view;
    }

    //Método manipulador pertencente a classe ButtonHandler
    @Override
    public void actionPerformed(ActionEvent arg0) {
           // método manipulador para tratar uma ação a ser executada por um componente gráfico de View
    }
}
```

Fique atento!

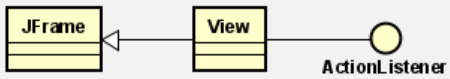
Observe que apenas a solução para o tratamento do evento por uma classe secundária não reconhece os componentes gráficos de uma *window* como variáveis globais, *claro*! Sendo necessária a passagem da view para a classe que tratará os eventos de seus componentes gráficos. Esta é uma problemática de comunicação entre classes.

Você Sabia?

Nas soluções apresentadas observamos o relacionamento entre classes e interfaces! Você saberia modelar cada uma delas? Vejamos as representações nos diagramas de classes abaixo:

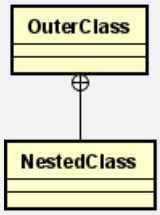
Classe com Herança e realização de interface:

```
class View extends JFrame implements ActionListener{}
```



Nested Classe:

```
class OuterClass {
    ...
    //além de default, o encapsulamento de NestedClass pode ser public, protected ou private
    class NestedClass {
        ...
    }
}
```

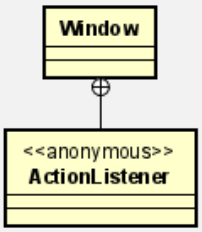


Atenção à terminologia: As classes aninhadas são divididas em duas categorias: não estáticas e estáticas. As classes aninhadas não estáticas são chamadas de classes internas (inner classes). As classes aninhadas que são declaradas estáticas são chamadas de classes aninhadas estáticas (static nested classes). Por exemplo:

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }
    static class StaticNestedClass {
        ...
    }
}
```

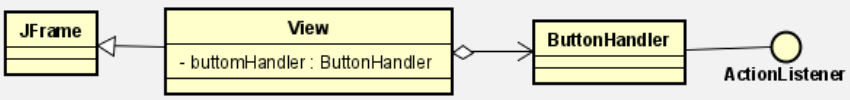
Anonymous inner classes:

```
public class Window {
       public void tratamento(){
              ActionListener l = new ActionListener() {
                     @Override
                     public void actionPerformed(ActionEvent e) {
                            // tratamento do evento
                     }
              };
       }
}
```



Atenção à terminologia: Não confundir "Anonymous Inner Class:" (classe interna anônima) com "Anonymous Bound Class" (classe vinculada anônima). Vejamos um exemplo de Anonymous Bound Class:

Comunicação entre classes com realização de interfaces:



2) Agora é hora de permitir que a tela de cadastro de usuário (**Fig. 1B**) tenha suas funcionalidades implementadas. Conforme o diagrama proposto por "O Furão", quando cliente está selecionado tem-se a opção para o recebimento de uma propaganda, mas quando é um funcionário (**Fig. 1C**) não há esta opção e deve-se dispor de campo para ser informada uma matrícula. Também deve ser codificada a funcionalidade para o botão Adicionar, ou seja, adicionar um novo usuário na base de dados (BaseDados.class). Escolha uma das abordagens apresentadas na seção "Relembrando!".

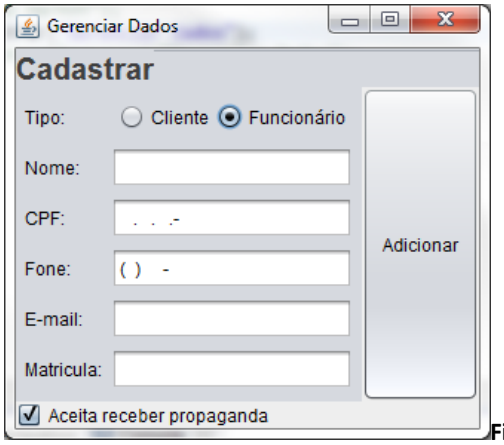


Fig. 1C

3) A partir das quatro abordagens apresentadas na seção "Relembrando", implemente a seguinte aplicação Java:

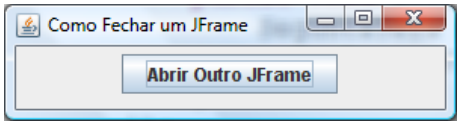


Fig. 2A

Ao clicar no botão "Abrir Outro JFrame" (tratamento de evento por classe interna anônima) irá carregar outro JFrame:

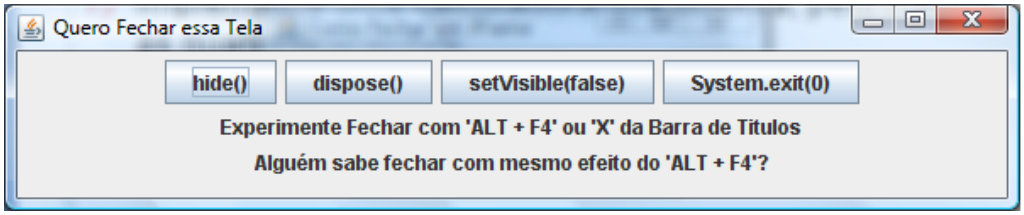


Fig. 2B

Observe as funcionalidades para os botões:

- hide() – dispara o método hide() fechando o segundo JFrame. Utilize tratamento de evento em classe realizando uma interface.
- dispose () – dispara o método dispose() fechando o segundo JFrame. Utilize tratamento de evento por classe interna.
- setVisible(false) – dispara o método setVisible() fechando o segundo JFrame. Utilize tratamento de evento por outra classe que não interna.
- System.exit(0) – encerra a aplicação. Utilize tratamento de evento por classe interna anônima.

(Opcional) Mão na Massa!

4) **(Opcional)** Implemente a GUI da calculadora ao lado (Fig. 3), mas devem ser utilizados exclusivamente as bibliotecas disponíveis no JDK, ou seja, em java e javax, sem o auxílio de recursos de IDE's do tipo *drag-and-drop*, como, por exemplo, a *palette de design* do Eclipse. Observe a disposição das opções da calculadora e escolha o devido layout. Realize os devidos tratamentos de eventos de maneira que tenha funcionalidade para pelo menos as quatro operações matemáticas. Lembre-se que as ações dos botões devem refletir a área de texto da calculadora, a qual não poderá ser editada pelo usuário. Adote a abordagem de "tratamento de evento por outra classe que não interna".

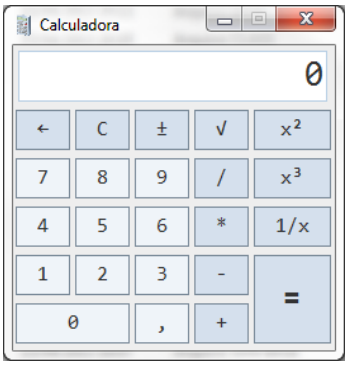


Fig. 3
(260 x 255)