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

http://ava.ufrpe.br/

https://sigs.ufrpe.br/sigaa/ava/index.jsf

**Disciplina:** Modelagem e Programação Orientada a Objetos (MPOO)

**Profº:** Richarlyson D'Emery

## LISTA DE EXERCÍCIOS IV

**Leia atentamente as instruções gerais:**

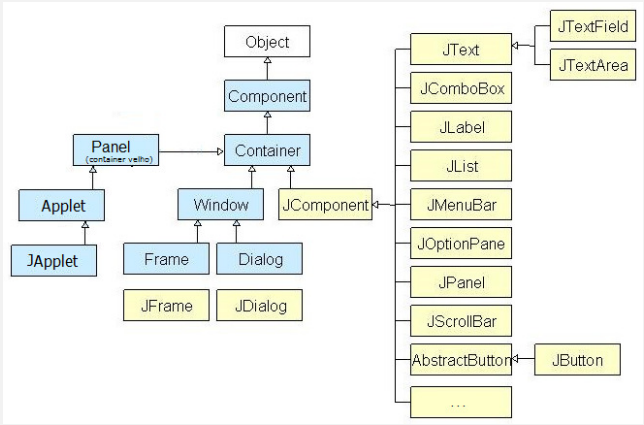
- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listaIV.SeuNomeSobrenome**, o qual deverá ter **pastas de pacotes** para cada questão: questao1, questao2, e assim sucessivamente, contendo todas as respostas da lista.
- A lista envolve questões práticas.

## Fique atento!

Prezado aluno, esta é a lista de exercícios é relativa ao assunto de "Componentes Gráficos" e "Tratamento de Eventos".

## Saiba Mais!

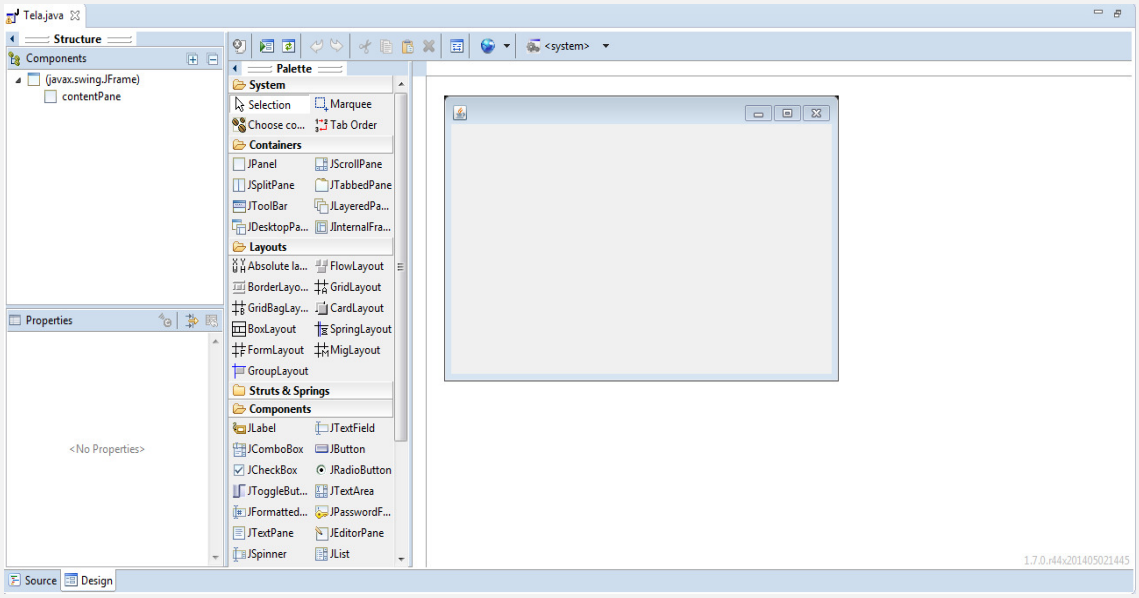
Os elementos básicos necessários para criar uma interface gráfica (GUI, do inglês *graphical user interface*) para um usuário utilizar um sistema estão em dois pacotes java.awt e javax.swing, em que awt foi o primeiro conjunto de classes Java para construir GUI's, enquanto swing é uma extensão de awt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

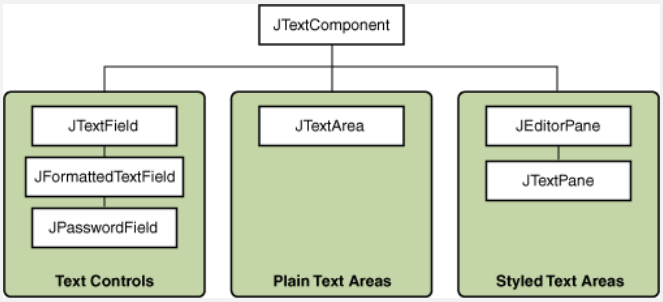
A aparência e a forma em que o usuário interage com a aplicação são chamados de look and feel da aplicação.

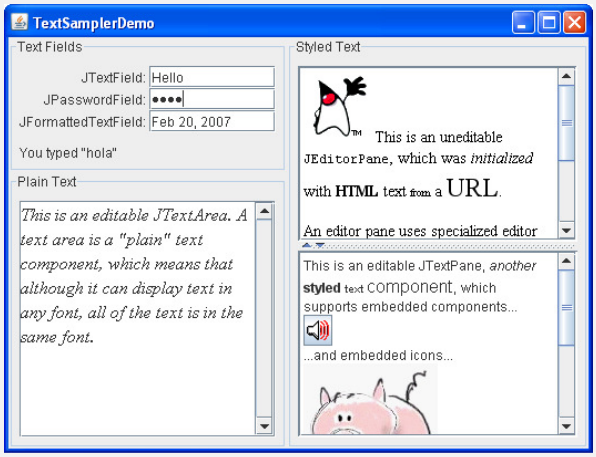
Desde a atualização 10 do Java SE 6, as GUI's passaram a ter uma *cara* nova, elegante e compatível com várias plataformas, conhecida como Nimbus.

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

## Saiba 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

## Relembrando!

Na aula de MPOO introduzimos o assunto de componentes gráficos em que aprendemos a criar interfaces gráficas para os nossos sistemas. Mas, para que o usuário possa interagir com uma GUI, é necessário realizar o devido Tratamento de Eventos para suas funcionalidades. Também vimos diversas possibilidades de codificação para a interação com as telas (Tratamento de Eventos!):

- 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
    }
}
```

*(continua na próxima página...)*

```java
//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!

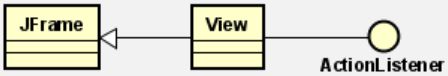
Observe que apenas a solução para o tratamento do evento por uma classe secundária não reconhece os componentes gráficos de uma *window* como variáveis globais, *claro*! Sendo necessária a passagem da view para a classe que tratará os eventos de seus componentes gráficos. Esta é uma problemática de comunicação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

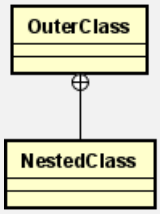
***Classe com Herança e realização de interface:***

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




***Nested Classe:***

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

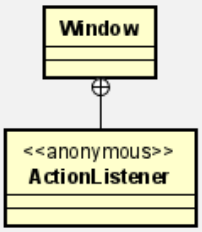



*Atenção à terminologia: As classes aninhadas são divididas em duas categorias: não estáticas e estáticas. As classes aninhadas não estáticas são chamadas de classes internas (inner classes). As classes aninhadas que são declaradas estáticas são chamadas de classes aninhadas estáticas (static nested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

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

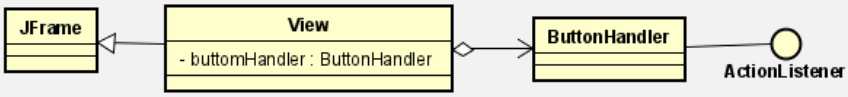
***Anonymous inner classes:***

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



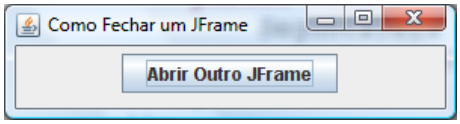
*Atenção à terminologia:* *Não confundir "Anonymous Inner Class:" (classe interna anônima) com "Anonymous Bound Class" (classe vinculada anônima). Vejamos um exemplo de Anonymous Bound Class:*


***Comunicação entre classes com realização de interfaces:***

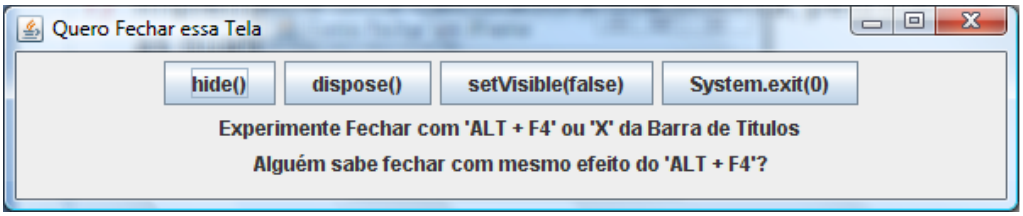



## Mão na Massa!

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




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



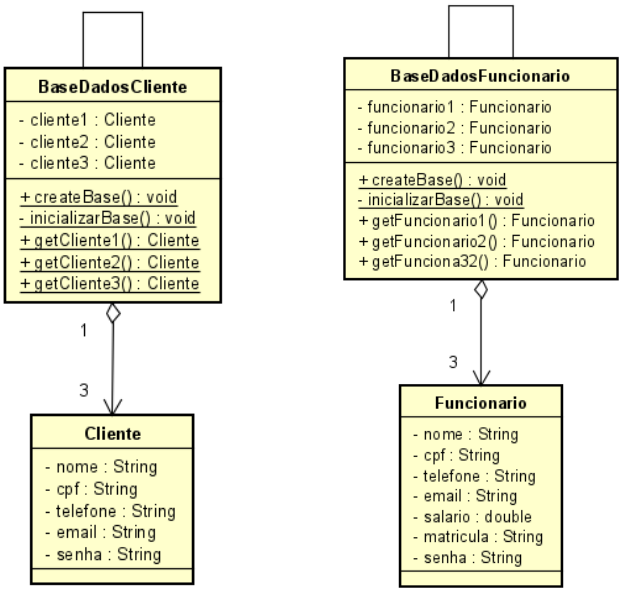

Observe as funcionalidades para os botões:

- 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.

## Desafio: Mão na Massa!

Você, aluno de MPOO, está experienciando situações-problemas do universo de desenvolvimento de software e começará a ser desafiado a solucionar problemas a partir de conhecimentos de Programação e Orientação a Objetos.


2) Um contratante solicitou a empresa MPOOSoftware LTDA a atualização de um sistema de cadastro. O Scrum Master de MPOOSoftware LTDA solicitou a um de seus programadores (de codinome *mustela putórius furo* – "O Furão") que resolvesse essa demanda. Para isso apresentou os seguintes diagramas de classes atuais da empresa:




São regras de negócios:

- RN01 – um cliente ou funcionário é identificado pelo seu cpf;
- RN02 – um cliente ou funcionário só é adicionado em uma base se tiver um cpf válido;
- RN03 – para entrar no sistema um cliente ou funcionário deverá informar seu login (email ou cpf) e senha;
- RN04 – a senha de um cliente ou funcionário deverá ser criada na primeira utilização do sistema;
- RN05 – a senha de um cliente ou funcionário deverá ter pelo menos 6 dígitos.

Assumindo o fato de:

- A base de clientes contém:
  - Cliente1: nome: “José Santos”, cpf: 941.860.760-30, telefone: (81) 99999-0000, e-mail: josesantos@gmail.com
  - Cliente2: nome: “Maria Silva”, cpf: 575.373.110-4830, telefone: (82) 99999-0001, e-mail: mariasilva@gmail.com
  - Cliente3: nome: “João Mamão”, cpf: 080.075.880-35, telefone: (87) 99999-0002, e-mail: jmamao@gmail.com
- A base de funcionários contém:
  - Funcionario1: nome: “Severino de Jesus”, cpf: 064.749.190-78, telefone: (81) 99999-1111, e-mail: sevjesus@gmail.com, salario: R$ 1500,00, matricula: “func001”
  - Funcionario2: nome: “Maria Silva”, cpf: 575.373.110-4830, telefone: (82) 99999-0001, e-mail: mariasilva@gmail.com, salario: R$ 1500,00, matricula: “func002”
  - Funcionario3: nome: “José Santos”, cpf: 941.860.760-30, telefone: (81) 99999-0000, e-mail: josesantos@gmail.com, salario: R$ 3000,00, matricula: “func001”

Codifique em Java as seguintes GUI’s do sistema:

- Para o usuário realizar o *login* no sistema deve informar seu login e senha (Fig. 1a). Caso algum campo não seja informado ao Entrar, devem-se exibir as informações para campo obrigatório (Fig. 1b ou Fig. 1c).

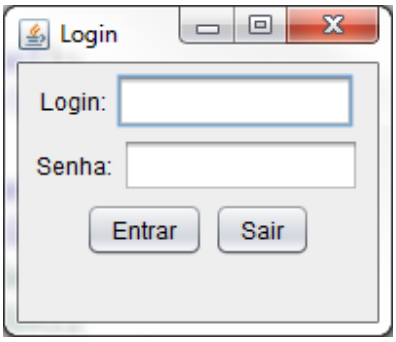



**Fig. 1a**

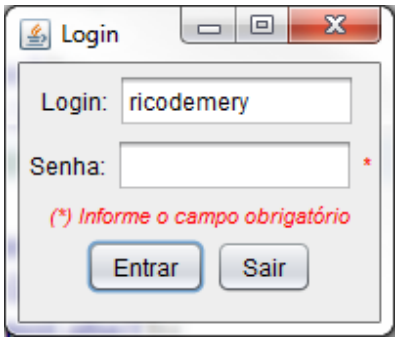



**Fig. 1b**

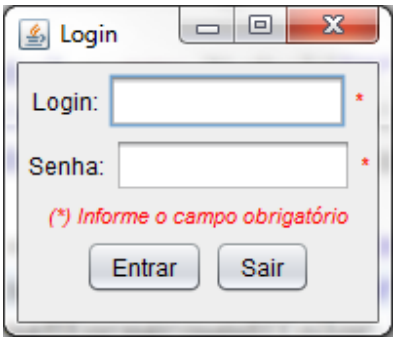



**Fig. 1c**

- Quando se tratar de primeiro acesso, o sistema deve tratar a regra de negócio RN04 através da GUI para cadastrar senha (Fig. 2a). Observe a necessidade de informar campo obrigatório quando login (Fi. 2b) ou login e senha (Fig. 2c) não é (são) informado(s) após clicar no botão Cadastrar. O campo login é automaticamente carregado a partir da tela Login (Fig. 2ª)

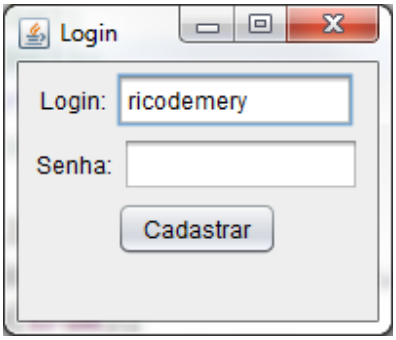



**Fig.2a**

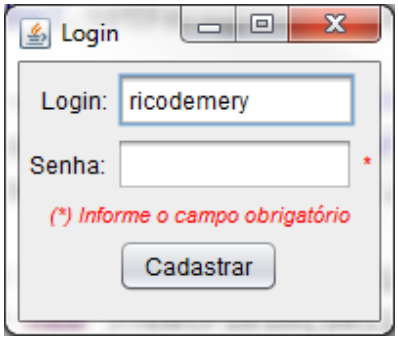



**Fig. 2b**

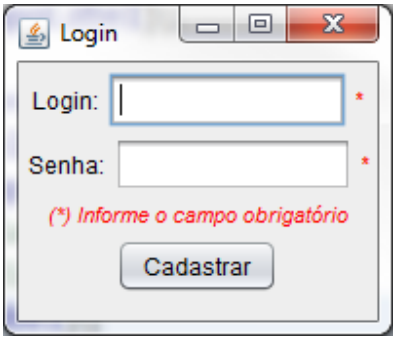
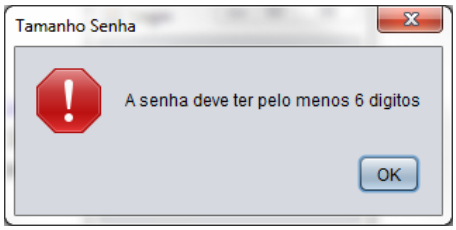
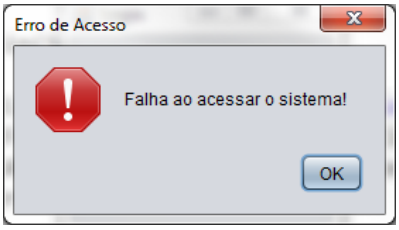



**Fig. 2c**

- Ao tentar cadastrar uma senha de maneira que a quantidade de dígitos seja inferior a 6, deve-se exibir a mensagem de erro:




- Quando um cliente ou funcionário não cadastrado tentar realizar o login ou cadastrar uma senha no primeiro acesso, o sistema deve exibir as mensagens de erro:




## Desafio!

3) Implemente em Java a GUI da calculadora ao lado (Fig. 3). Você deve utilizar componentes gráficos de javax.swing. 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 classe realizando a interface ActionListener".

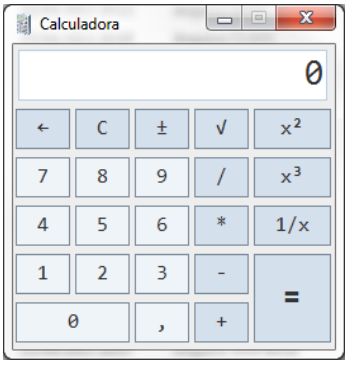



**Fig. 3**
*(260 x 255)*