

Site: <https://sigs.ufrpe.br/sigaa/ava/index.jsf>

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery

LISTA DE EXERCÍCIOS VIII

Leia atentamente as instruções gerais:

- No Eclipse crie um novo projeto chamado **br.edu.mpoo.listaVIII.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!

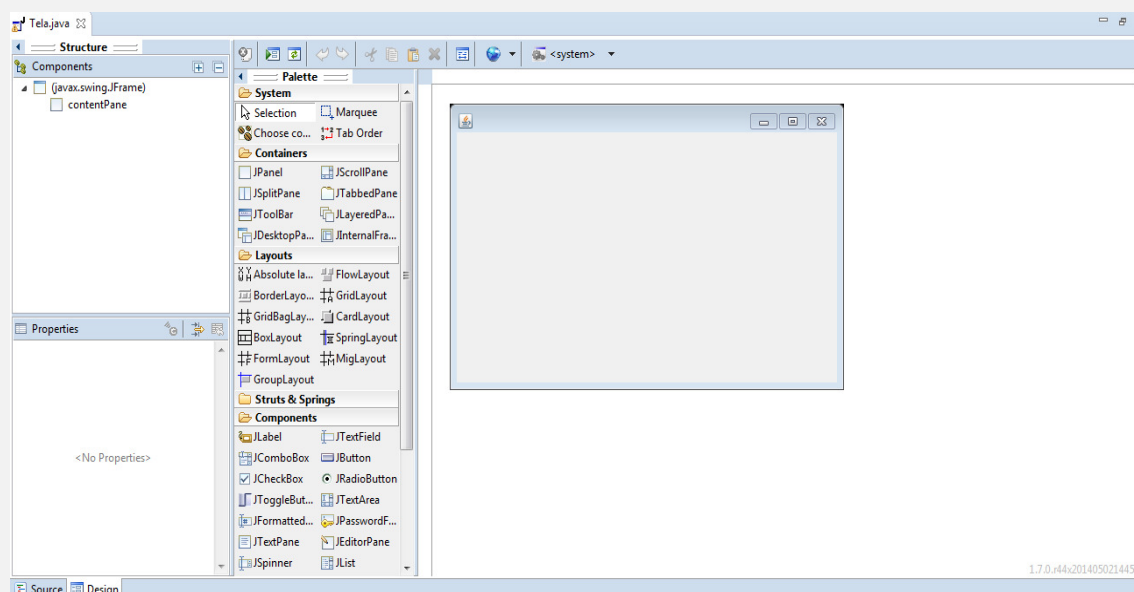
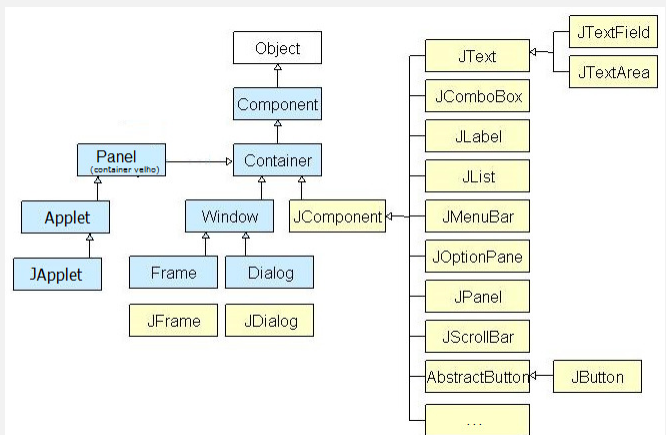
Os elementos básicos necessários para criar uma interface gráfica (GUI, do inglês *graphical user interface*) para um usuário utilizar um sistema estão em dois pacotes **java.awt** e **javax.swing**, em que **awt** foi o primeiro conjunto de classes Java para construir GUI's, enquanto **swing** é uma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

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

Os componentes podem ser atômicos (que não contém outros componentes gráficos) e complexos (voltados a processos específicos como, por exemplo, grande quantidade de informações, usar nós/diretórios, entre outros).

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.



Fique por dentro!

São classes de `javax.swing`:

- **Containers**

- `JFrame` – É a janela do software, o contêiner principal
- `JDialog` – Uma janela do tipo `Dialog Window`, também pode ser um container principal.
- `JPanel` – Permite a criação de painéis independentes de onde são armazenados outros componentes.
- `JScrollPane` – Permite vincular barras de rolagem em um contêiner.
- `JSplitPane` – Permite a criação de um container dividido em 2 seções.
- `JTabbedPane` – Permite a criação de abas, cada aba representa um container independente.
- `JDesktopPane` – Permite criar janelas dentro de uma janela principal.
- `JToolBar` – Permite entrar em uma barra de ferramentas.

- **Componentes Atômicos**

- `JLabel` – Permite vincular rótulos (seja texto ou imagem).
- `JButton` – Permite vincular botões simples.
- `JCheckBox` – Permite a criação de caixas de seleção, ideais para seleção individual.
- `JRadioButton` – Permite apresentar opções para seleção excludente.
- `JToggleButton` – Botão que quando pressionado permanecerá pressionado até que outro evento seja executado.
- `JComboBox` – Permite exibir uma lista de opções.
- `JScrollBar` – Permite exibir uma barra de rolagem, frequentemente usada em áreas de texto, lista de opções (combo ou list) ou painéis onde o conteúdo é maior que o tamanho do componente.
- `JSeparator` – Permite separar opções, é uma barra simples.
- `JSlider` – Permite vincular um *slider* em uma janela.
- `JSpinner` – Permite vincular uma caixa de texto com botões integrados para selecionar algum valor.
- `JProgressBar` – Define uma barra de progresso.

- **Componente de diálogo**

- `JOptionPane` – Permite exibir uma caixa de diálogo personalizável (um *feedback* para o usuário saber ou confirmar um ação).

- **Componentes de texto.**

- `TextField` – Permite inserir um campo de texto simples.
- `FormattedTextField` – Permite inserir campo de texto formatado. Utiliza uma máscara pré-definida (Ex.: data e cpf).
- `PasswordField` – Campo de texto que oculta os caracteres inseridos.
- `TextArea` – Permite vincular uma área de texto onde o usuário irá inserir informações ou simplesmente apresentar *strings* de texto.
- `EditorPane` – Permite vincular uma área de texto com propriedades de formatação.
- `TextPane` – Semelhante ao anterior, permitindo outras opções de formato, cores, ícones, entre outros.

- **Componentes do menu.**

- `MenuBar` – Permite vincular uma barra de menu.
- `Menu` – Permite vincular botões ou links que, quando pressionados, exibem um menu principal.
- `MenuItem` – Botão ou opção encontrada em um menu.
- `CheckboxMenuItem` – Item de menu como opções de caixa de seleção.
- `RadioButtonMenuItem` – Item de menu como botão de opção.
- `PopupMenu` – Opções do menu pop-up.

- **Componentes Complexos**

- `JTable` – Permite vincular uma tabela de dados com suas respectivas linhas e colunas.
- `JTree` – Carrega uma árvore onde é estabelecida uma determinada hierarquia visual, tipo de diretório.
- `JList` – Permite carregar uma lista de elementos, dependendo das propriedades você pode ter uma lista de seleção múltipla.
- `FileChooser` – É um componente que permite a busca e seleção de arquivos, entre outros.
- `ColorChooser` – Componente que permite carregar um painel seletor de cores

Você Sabia?

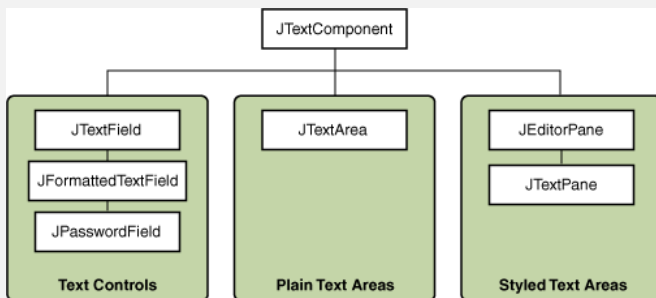
A aparência e a forma em que o usuário interage com a aplicação são chamados de **look and feel** da aplicação.

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**.

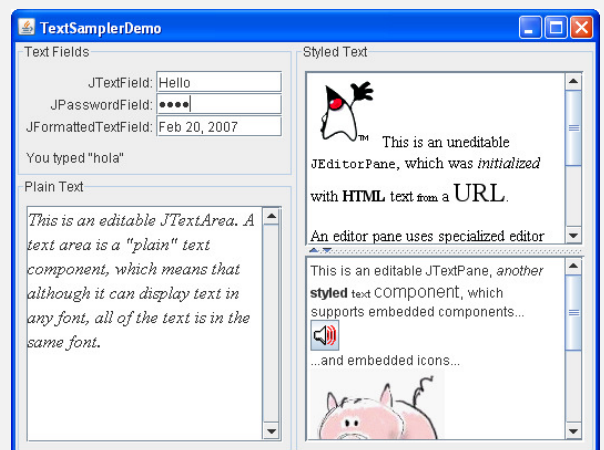
```
try {
    UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
} catch (ClassNotFoundException | InstantiationException
        | IllegalAccessException | UnsupportedLookAndFeelException e) {
    e.printStackTrace();
}
```

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>



Vamos Praticar! Criação de front-end!

1) As GUI's abaixo ilustram possibilidade de telas em que o usuário irá interagir com um sistema. Para o desenvolvimento do *front-end*, deve-se utilizar exclusivamente as bibliotecas disponíveis no JDK, ou seja, em **java** e **javax**, mas sem o auxílio de recursos de IDE's do tipo *drag-and-drop*, como, por exemplo, a *palette de design* do Eclipse. Faça uso do gerenciador de layout **FlowLayout**.

1.1) Crie a GUI para login de um sistema (Fig. 1a). Observe que para o campo senha o usuário não deverá ver o seu conteúdo, logo utilize **JPasswordField** (Veja a seção **Saiba Mais!**)

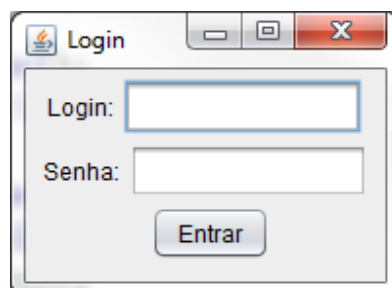


Fig. 1a. Tela Login

Como você faria para permitir o usuário trocar entre espiar a senha ou deixá-la de forma não explícita? Reflita seu pensamento na GUI.

- 1.2) Crie a GUI para o cadastro de um usuário em um sistema (Fig. 1b). Observe que o campo CPF não possui uma indicação de seu conteúdo. Pesquise como poderia melhorar a interface de maneira a colocar uma máscara. (Dica: MaskFormatter).

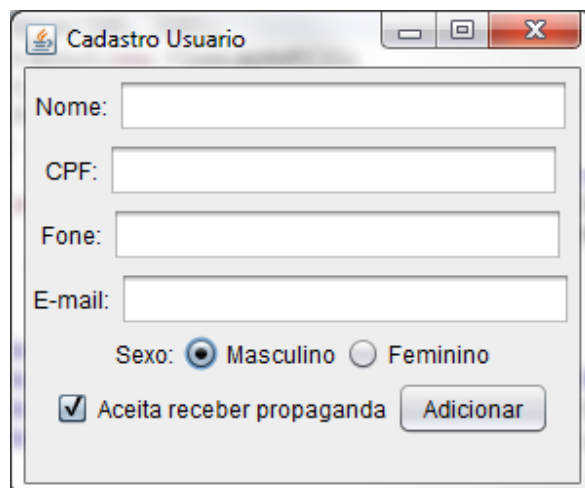


Fig. 1b. Tela Cadastrar Usuário

Fique Atento!

Observe a utilização do componente `JRadioButton`. Para que a seleção entre os tipos de sexo “Masculino” e “Feminino” 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`.



Relembrando!

Mas e essas GUI's tem funcionalidade? Se o usuário clicar em seus botões acontecerá algo?

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.
    }
}
```

(continua na próxima página...)

- 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...)

```
//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!

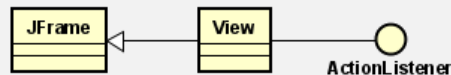
Observe que apenas a solução para o tratamento do evento por uma classe secundária não reconhece os componentes gráficos de uma *window* como variáveis globais, *claro!* Sendo necessária a passagem da view para a classe que tratará os eventos de seus componentes gráficos. Esta é uma problemática de comunicação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

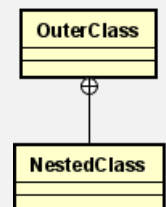
Classe com Herança e realização de interface:

```
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 {
        ...
    }
}
```

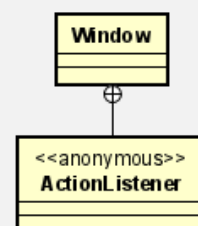


Atenção à terminologia: As classes aninhadas são divididas em duas categorias: não estáticas e estáticas. As classes aninhadas não estáticas são chamadas de classes internas (inner classes). As classes aninhadas que são declaradas estáticas são chamadas de classes aninhadas estáticas (static nested classes).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

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

Anonymous inner classes:

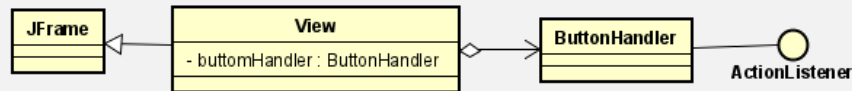
```
public class Window {
    public void tratamento(){
        ActionListener l = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // tratamento do evento
            }
        };
    }
}
```



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:



Mão na Massa!

2) A partir das quatro abordagens apresentadas na seção “Relembrando”, implemente a seguinte aplicação Java:

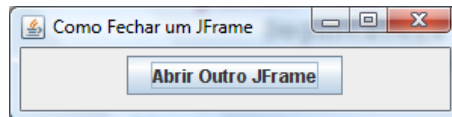


Fig. 2a. Tela Inicial

Ao clicar no botão “Abrir Outro JFrame” (Fig.2a) (tratamento de evento por classe interna anônima) irá carregar outro JFrame:

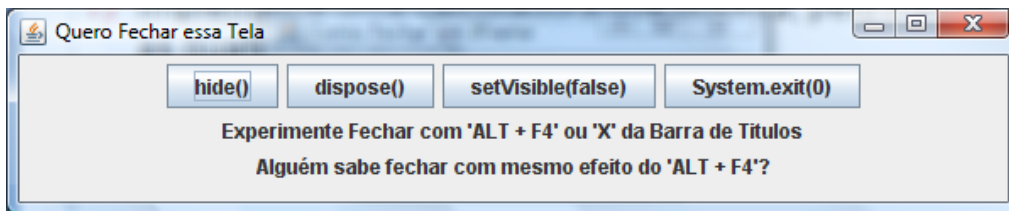


Fig. 2b. Tela Secundária

Observe as funcionalidades para os botões:

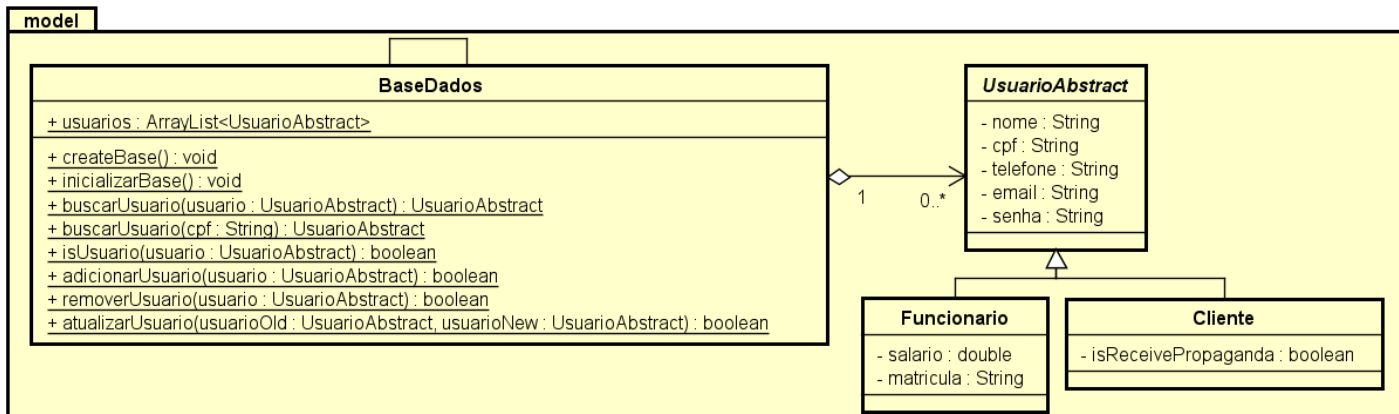
- `hide()` – dispara o método `hide()` fechando o segundo JFrame. Utilize tratamento de evento em classe realizando uma interface.
- `dispose()` – dispara o método `dispose()` fechando o segundo JFrame. Utilize tratamento de evento por classe interna.
- `setVisible(false)` – dispara o método `setVisible()` 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 – front-end vs back-end!

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



- 3) Na **Lista de Exercícios VII**, vimos a demanda de um contratante de um sistema de cadastro para a empresa MPOOSoftware LTDA. O Scrum Master de MPOOSoftware LTDA solicitou a um de seus programadores (de codinome *mustela putórus furo* – “O Furão”) que agora implementasse uma GUI para que o contratante estivesse mais familiarizado com o sistema. Para isso apresentou além do diagrama de classe, que possui uma base de dados que utiliza o conceito de polimorfismo de objetos, um *mockup* (Fig. 3) de tela. Sendo assim codifique a GUI 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.



São regras de negócios:

- RN01 – um cliente ou funcionário é identificado pelo seu cpf;
- RN02 – um cliente ou funcionário só é adicionado a base se tiver um cpf válido;
- RN03 – um cliente ou funcionário só poderá ser cadastrado uma única vez;
- RN04 – para entrar no sistema um cliente ou funcionário deverá informar seu login (email ou cpf) e senha;
- **RN05 – a senha de um cliente ou funcionário deverá ser criada na primeira utilização do sistema, com isso a senha default deverá ser 123456;**
- RN06 – a senha de um cliente ou funcionário deverá ter pelo menos 6 dígitos;
- RN07 – a codificação deve aproveitar comportamentos já definidos, evitando a duplicidade de programação; e
- RN08 – a empresa só envia propaganda se o cliente permitir recebê-la.

Assumindo o fato de:

- A base conter os clientes:
 - 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”

Observe a existência de cpf's inválidos de maneira proposital e que não deve violar a RN02.

É GUI de cadastro:

Fig. 3. Mockup da Tela de Cadastro

3.1) Observe a opção para “Aceita receber propaganda”, onde essa informação deverá ser armazenada? Reflita sua solução na codificação do sistema.

As opções do *mockup* ilustrado estão relacionadas à opção selecionada “Cliente”, portanto, quando a opção “Funcionário” for selecionada, a GUI deve ser atualizada para que os campos relativos aos atributos de um funcionário sejam exibidos. A ação de adicionar deve respeitar as regras de negócios: RN01, RN02, RN03, RN05 e RN07.

- 4) Podemos observar que a GUI de cadastro da questão anterior não há campo para inserção de senha. E a **RN05 destaca** que o atributo senha só será atualizado na primeira utilização de login do sistema.

Implemente as novas GUI's para o login de um usuário e atualização de senha quando se tratar de primeiro acesso:

- Para o usuário realizar o *login* no sistema deve informar seu login e senha (RN04) (Fig. 4a). Caso algum campo não seja informado ao Entrar, devem-se exibir as informações para campo obrigatório (Fig. 4b ou Fig. 4c).

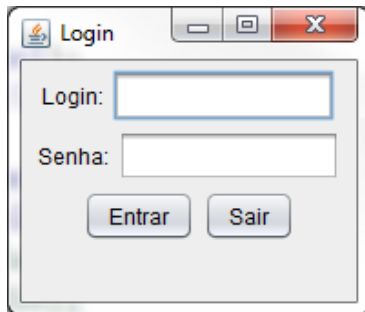


Fig.4a

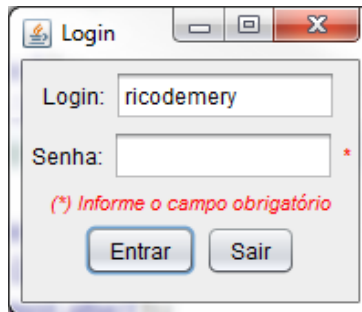


Fig. 4b

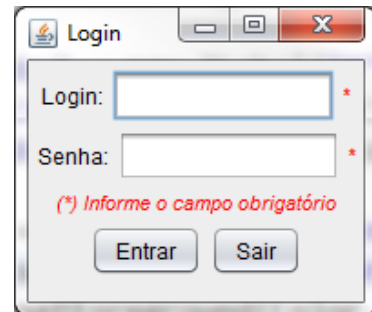


Fig. 4c

- Quando se tratar de primeiro acesso, o sistema deve tratar a regra de negócio RN04 através da GUI para cadastrar senha (Fig. 4d). Observe a necessidade de informar campo obrigatório quando login (Fig. 4e) ou login e senha (Fig. 4f) não é (são) informado(s) após clicar no botão Cadastrar. O campo login é automaticamente carregado a partir da tela Login (Fig. 4a)

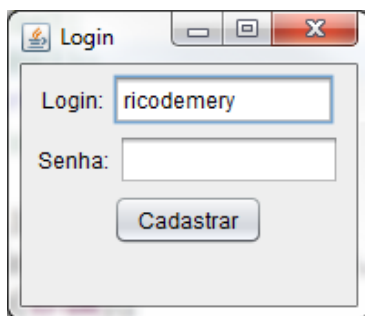


Fig.4d

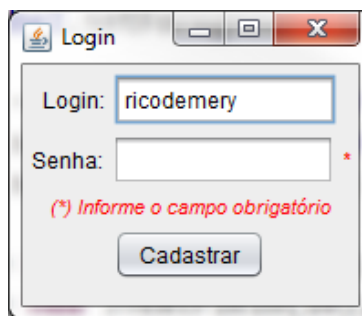


Fig. 4e

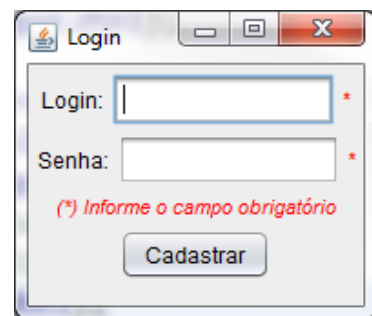
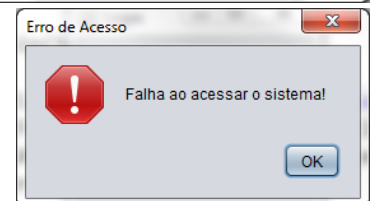
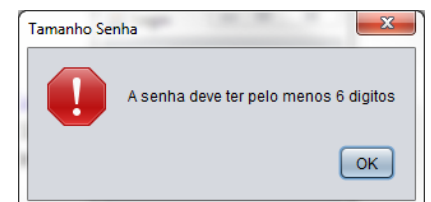


Fig. 4f

- 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!

- 5) Implemente em Java a GUI da calculadora ao lado (Fig. 5). 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".

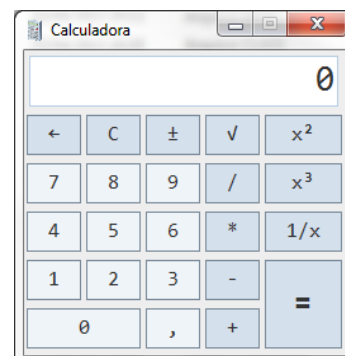


Fig. 5
(260 x 255)