

Interfaces gráficas com JAVA.

Assim como a maioria das linguagens de programação modernas o JAVA implementa interfaces gráficas para uma melhor experiência de uso por parte dos usuários.

Não diferente de todos os outros recursos interfaces gráficas no JAVA se organizam dentro de pacote, são dois o **AWT (Abstract Window Toolkit)** e a **“SWING” (Widget Toolkit)**.

AWT – Foi o primeiro pacote gráfico do JAVA até ser substituído pelo pacote **“SWING”** apesar de não ser mais recomendado usar o **“AWT”** para desenvolver interfaces gráficas esse ainda é usado para tratar eventos associados aos componentes visuais, inclusive do pacote **“SWING”**.

SWING – É o pacote mais atual e substitui o **“AWT”** sua aparência é bem mais atraente que a do pacote **“AWT”** além de ser mais compatível com os gerenciadores de janelas dos diversos sistemas operacionais que suportam o JAVA.

Existem duas formas de se trabalhar com interfaces gráficas no JAVA, podemos programar manualmente o que só é interessante para aprender melhor a estrutura dos componentes e seus comportamentos ou podemos construir interfaces através de um **“IDE”** como o **“NetBeans”** ou o **“Eclipse”**, por exemplo. Veremos as duas formas nesse curso e no que diz respeito a segunda forma vamos usar o **“NetBeans”** como vem sendo feito até esse momento do curso.

Gerenciadores de layout.

São três classes do JAVA responsáveis por gerenciar o posicionamento dos elementos gráficos em um formulário, são elas:

BorderLayout – Esse gerenciador de layout organiza os elementos gráficos de um formulário em quatro posições, **NORTH (TOPO)**, **SOUTH (SUL)**, **EAST (ESQUERDO)**, **WEST (DIREITO)** e **CENTER (CENTRO)**.

FlowLayout – Posiciona os elementos da esquerda para a direita enquanto houver espaço na horizontal quando esse acaba posiciona o próximo elemento abaixo de seus antecessores e começa uma nova **“linha”**.

GridLayout – Organiza os elementos gráficos em um formulário através de linhas e colunas como uma tabela ou grade.

Observação: Os gerenciadores de layout citados acima podem ser aplicados tanto no pacote **“AWT”** quanto no **“SWING”**.

Exemplo de BorderLayout usando a classe AWT:



Para exemplificar cada uma das posições foram usados botões com rótulos que indicam seu posicionamento quando uma posição não é usada outra vai tomar seu lugar.

A imagem abaixo ilustra o código da janela acima:

```
1 package gui;
2
3 import java.awt.BorderLayout;
4 import java.awt.Button;
5 import java.awt.Frame;
6
7 public class BorderLayoutAWT {
8
9     public static void main(String[] args) {
10
11         //A classe frame cria uma janela capaz de comportar uma aplicação.
12         //Por padrão será usado o gerenciador de layout BorderLayout
13         Frame janela = new Frame();
14
15         //O método setTitle define um título para a janela, isso poderia se feito
16         //também pelo construtor padrão da classe Frame.
17         janela.setTitle("Título da janela.");
18
19         //A classe button permite a criação de botões, podemos adicionar um
20         //rótulo ao botão passando um valor String para o construtor padrão da
21         //classe button
22         Button btnNorte = new Button("Botão Norte");
23         Button btnSul = new Button("Botão Sul");
24         Button btnLeste = new Button("Botão Leste");
25         Button btnOeste = new Button("Botão oeste");
26         Button btnCentro = new Button("Botão Centro");
27
28         //Após criar o botão devemos adiciona-lo ao objeto da janela através do método
29         //add.
30         janela.add(btnNorte, BorderLayout.NORTH);
31         janela.add(btnSul, BorderLayout.SOUTH);
32         janela.add(btnLeste, BorderLayout.EAST);
33         janela.add(btnOeste, BorderLayout.WEST);
34         janela.add(btnCentro, BorderLayout.CENTER);
35
36         /*É interessante que os métodos de setSize e setVisible sejam os últimos
37         do código logo após a definição de todos os elementos gráficos da tela.
38         */
39
40         //O método setSize permite definir o tamanho da janela.
41         janela.setSize(640, 480);
42
43         //Torna a janela visível.
44         janela.setVisible(true);
45     }
46 }
47
48 }
```

Como é apenas um exemplo a classe que implementa o formulário e seus componentes faz também o papel de classe principal.

Elementos gráficos usados nessa tela:

Frame – Cria um formulário onde os elementos gráficos serão posicionados.

Button – Cria botões.

Observe que tudo começa com a criação de um objeto da classe “Frame” na linha treze (13) que é o mesmo que um formulário é esse objeto que vai “carregar” os elementos gráficos da tela nesse caso os botões que representam as posições do “BorderLayout”.

A linha dezessete (17) configura um valor para o título da janela através do método “setTilte” da classe “**Frame**”.

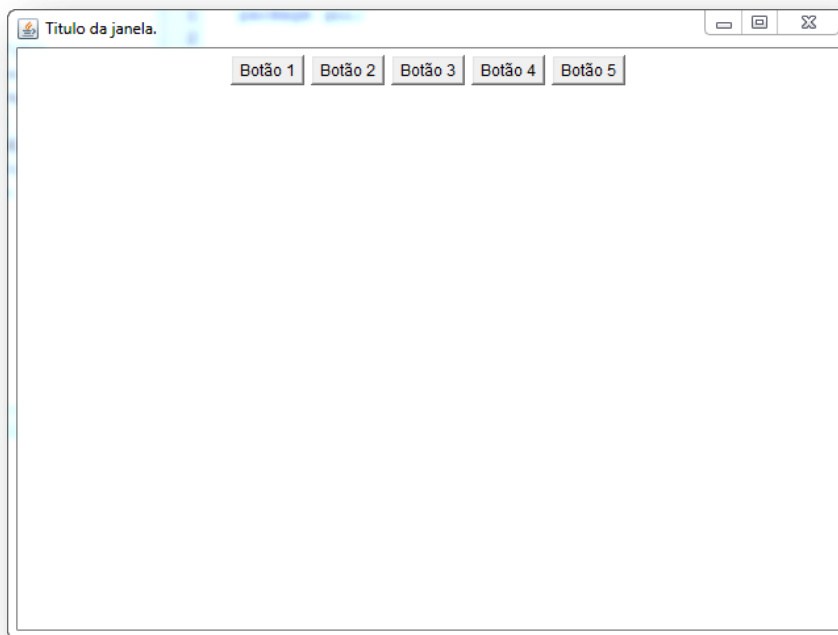
Das linhas vinte e dois (22) a vinte e seis (26) São criados objetos da classe “**Button**”, observe que o nome de cada botão é configurado através do construtor da classe “**Button**”.

Da linha trinta (30) a trinta e quatro (34) temos uma das ações mais importantes desse código que é o processo de adicionar os elementos gráficos nesse caso os botões ao “**Frame**” através do método “add” do objeto da classe “**Frame**”, observe que são passados dois parâmetros o primeiro é o elemento gráfico em si e o segundo é a posição que esse elemento vai ocupar no “**Frame**”.

A linha quarenta e um (41) configura o tamanho do “**Frame**” através do método “**setSize**” do objeto dessa mesma classe.

E a linha quarenta e quatro (44) é com certeza a mais importante de todas até aqui pois essa linha torna o “**Frame**” visível, por padrão seu valor é falso.

Exemplo de FlowLayout usando a classe AWT:



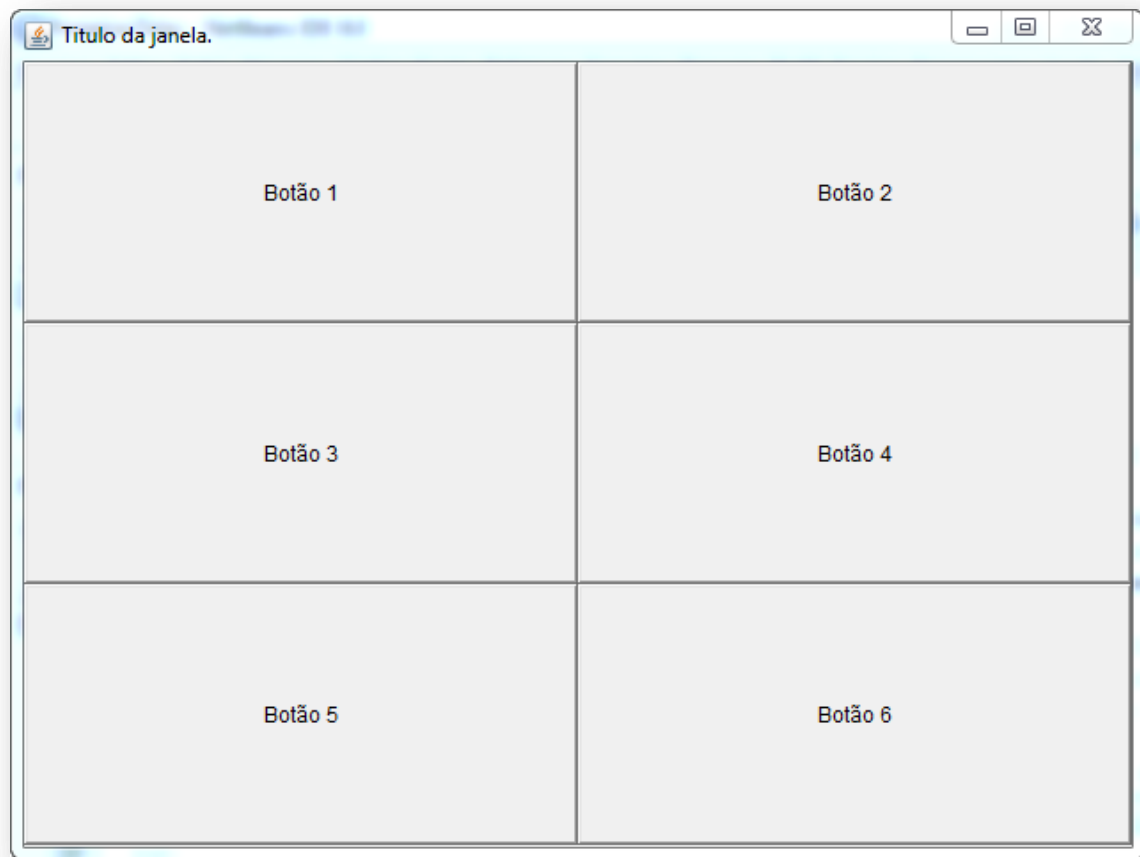
Observe a disposição horizontal dos botões essa é a principal característica do gerenciador de layout “**FlowLayout**”.

A imagem abaixo ilustra o código da janela acima:

```
1 package gui;
2
3 import java.awt.Button;
4 import java.awt.FlowLayout;
5 import java.awt.Frame;
6
7 public class UsandoFlowLayout {
8
9     public static void main(String[] args) {
10
11         //A classe frame cria uma janela capaz de comportar uma aplicação.
12         Frame janela = new Frame();
13
14         //O método setTitle define um título para a janela, isso poderia se feito
15         //também pelo construtor padrão da classe Frame.
16         janela.setTitle("Título da janela.");
17
18         //Define o gerenciador de layout que será usado pela classe Frame.
19         janela.setLayout(new FlowLayout());
20
21         //A classe button permite a criação de botões, podemos adicionar um
22         //rótulo ao botão passando um valor String para o construtor padrão da
23         //classe button
24         Button btn1 = new Button("Botão 1");
25         Button btn2 = new Button("Botão 2");
26         Button btn3 = new Button("Botão 3");
27         Button btn4 = new Button("Botão 4");
28         Button btn5 = new Button("Botão 5");
29
30         //Adicionando botões em gerenciador de layout FlowLayout, observe que
31         //nesse gerenciador não existe a necessidade de informar uma posição.
32         janela.add(btn1);
33         janela.add(btn2);
34         janela.add(btn3);
35         janela.add(btn4);
36         janela.add(btn5);
37
38         /*É INTERESSANTE QUE OS MÉTODOS DE setSize e setVisible sejam os últimos
39         do código logo após a definição de todos os elementos gráficos da tela.
40         */
41
42         //O métodop setSize permite definir o tamanho da janela.
43         janela.setSize(640, 480);
44
45         //Torna janela visível.
46         janela.setVisible(true);
47
48     }
49
50 }
```

Analizando o código acima não temos muitas novidades se comparado com o anterior na verdade somente uma linha foi adicionada e nos interessa nesse momento a linha dezenove (19) “**janela.setLayout(new FlowLayout());**” onde é informado ao objeto de “Frame” qual vai ser o gerenciador de layout usado, nesse caso o “**FlowLayout**”. Na tela anterior isso não foi feito porque o “**BorderLayout**” é o gerenciador de layout padrão no JAVA.

Exemplo de GridLayout usando a classe AWT:



Observe a disposição em linhas e colunas característica do gerenciador de layout “**GridLayout**”.

A imagem abaixo ilustra o código da janela acima:

```
1 package gui;
2
3 import java.awt.Button;
4 import java.awt.Frame;
5 import java.awt.GridLayout;
6
7 public class UsandoGridLayout {
8
9     public static void main(String[] args) {
10         //A classe frame cria uma janela capaz de comportar uma aplicação.
11         Frame janela = new Frame();
12
13         //O método setTitle define um título para a janela, isso poderia se feito
14         //também pelo construtor padrão da classe Frame.
15         janela.setTitle("Título da janela.");
16
17         //Define o gerenciador de layout que será usado pela classe Frame.
18         janela.setLayout(new GridLayout(3,2));
19
20         //A classe button permite a criação de botões, podemos adicionar um
21         //rótulo ao botão passando um valor String para o construtor padrão da
22         //classe button
23         Button btn1 = new Button("Botão 1");
24         Button btn2 = new Button("Botão 2");
25         Button btn3 = new Button("Botão 3");
26         Button btn4 = new Button("Botão 4");
27         Button btn5 = new Button("Botão 5");
28         Button btn6 = new Button("Botão 6");
29
30         //Adicionando botões em gerenciador de layout GridLayout, observe que
31         //nesse gerenciador os componentes gráficos vão ocupar as "celulas" na
32         //medida em vão sendo adicionados ao frame (janela).
33         janela.add(btn1);
34         janela.add(btn2);
35         janela.add(btn3);
36         janela.add(btn4);
37         janela.add(btn5);
38         janela.add(btn6);
39
40         /*É INTERESSANTE QUE OS MÉTODOS DE setSize e setVisible sejam os últimos
41         do código logo após a definição de todos os elementos gráficos da tela.
42         */
43
44         //O método setSize permite definir o tamanho da janela.
45         janela.setSize(640, 480);
46
47         //Torna janela visível.
48         janela.setVisible(true);
49
50     }
51
52 }
```

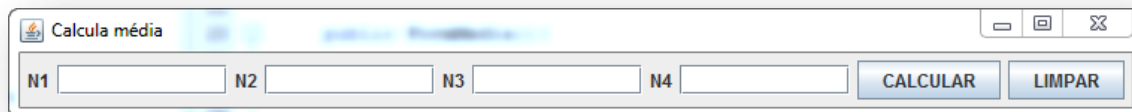
Novamente boa parte do código é muito semelhante com os vistos anteriormente então vou comentar somente a linha relevante, observe a linha dezoito (18) onde é configurado o gerenciador de layout “**janela.setLayout(new GridLayout(3,2));**”, note que “**GridLayout**” requer dois parâmetros um para linhas e outro para colunas respectivamente.

Desenvolvendo um formulário com SWING.

Agora que temos uma noção maior de como interfaces gráficas funcionam no JAVA vamos deixar o pacote “**AWT**” um pouco de lado e partir para o pacote “**SWING**” que tem um funcionamento muito parecido salvo algumas pequenas

diferenças como a forma que adicionamos os elementos gráficos e a aparência dos componentes que é bem mais atraente.

Vamos construir um pequeno formulário que vai calcular a média de quatro números. A imagem abaixo ilustra o formulário:

A screenshot of a Windows application window titled 'Calcula média'. The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. Inside the window, there are four text input fields labeled 'N1', 'N2', 'N3', and 'N4' arranged horizontally. To the right of these fields are two buttons: 'CALCULAR' and 'LIMPAR'.

A imagem abaixo ilustra o código da janela acima:


```
2 package calculamedia;
3
4 import java.awt.Container;
5 import java.awt.FlowLayout;
6 import javax.swing.JButton;
7 import javax.swing.JFrame;
8 import javax.swing.JLabel;
9 import javax.swing.JMenu;
10 import javax.swing.JTextField;
11
12 public class FormMedia extends JFrame{
13
14     private JButton btnMedia;
15     private JButton btnLimpar;
16     private JTextField txtN1;
17     private JTextField txtN2;
18     private JTextField txtN3;
19     private JTextField txtN4;
20     private JLabel lblMedia;
21     private Container container;
22
23     public FormMedia(){
24
25         super("Calcula média");
26
27         this.setLayout(new FlowLayout());
28
29         container = this.getContentPane();
30
31         btnMedia = new JButton("CALCULAR");
32         btnLimpar = new JButton("LIMPAR");
33
34         txtN1 = new JTextField(10);
35         txtN2 = new JTextField(10);
36         txtN3 = new JTextField(10);
37         txtN4 = new JTextField(10);
38
39         lblMedia = new JLabel("0.0");
40
41         container.add(new JLabel("N1"));
42         container.add(txtN1);
43
44         container.add(new JLabel("N2"));
45         container.add(txtN2);
46
47         container.add(new JLabel("N3"));
48         container.add(txtN3);
49
50         container.add(new JLabel("N4"));
51         container.add(txtN4);
52
53         container.add(btnMedia);
54         container.add(btnLimpar);
55
56         this.pack();
57
58         this.setVisible(true);
59
60     }
61 }
62 }
```

Note que foi feita uma herança entre a classe “**FormMedia**” e a classe “**JFrame**” na linha doze (12). Uma outra coisa a ser notada aqui é o fato de que todo componente visual do pacote “**SWING**” começa com a letra “J” seguido pelo nome do componente.

Na linha vinte e três declaramos um método construtor para a classe “**FormMedia**” e que todo o formulário está contido nesse método, logo quando for instanciado um objeto da classe o formulário vai ser construído automaticamente essa técnica é muito comum em classe que representam formulários.

A linha vinte e sete (27) configura o gerenciador de layout da classe como “**FlowLayout**”.

Das linhas trinta e quatro (34) até trinta e sete (37) são instanciados os objetos das caixas de texto, observe que é passado o número dez (10) para o construtor de cada classe **"JTextField"**, isso define o tamanho visível de cada campo de texto mas não limita os campos a dez caracteres.

A linha trinta e nove (39) cria a **"JLabel"** que vai receber o valor do cálculo da média.

Das linhas quarenta e um (41) a cinquenta e quatro (54) são adicionadas as **"JLabel"** e os **"JTextField"** respectivamente pois os elementos gráficos aparecem no **"JFrame"** na ordem em que são adicionados, observe a linha quarenta e um (41) **"container.add(new JLabel("N1"))"** onde é criado um objeto de **"JLabel"** no momento da adição desse ao container, diferente do **"JLabel"** que vai armazenar o resultado esse não possui um atributo, isso se deve ao fato de que não vamos manipular mais esse **"JLabel"** depois de criado então para economizar linhas de código e criar uma classe mais concisa e de fácil leitura e manutenção usa-se essa técnica o nome que se dá a esse tipo de objeto é **"objeto anônimo"** essa técnica se repete nas linhas 44, 47 e 50 para todos os **"JLabel"** que vão indicar o número a ser digitado.

A linha cinquenta e seis (56) chama o método **"pack"** de **"JFrame"** que serve para configurar o tamanho do formulário de forma que acomode todos os elementos visuais sem excessos.

Adicionando tratamento de eventos no formulário de cálculo de média.

Agora vamos programar os botões **"CALCULAR"** e **"LIMPAR"** para isso precisamos entender o conceito de eventos dentro do JAVA.

Um evento é uma ação que um elemento gráfico sofre, essas ações geralmente são disparadas pelo usuário do sistema mas podem vir de outras fontes como um outro programa ou do sistema operacional.

O exemplo mais óbvio de evento é quando um usuário clica com o mouse sobre um botão no JAVA todo elemento gráfico tem um evento padrão o do botão por um acaso é o clique, sempre que um elemento gráfico sofre uma ação de evento é gerado um objeto do tipo **"ActionEvent"** esse evento de ação é capturado por um **"Listener"** ou **"ouvinte"** é um recurso implícito dentro JAVA que fica **"escutando"** todas as ações de evento, logo toda vez que uma ação dessa ocorre é possível recuperá-la e tratá-la através de um método chamado **"actionPerformed"** que captura automaticamente cada ação de evento disparada contra um elemento gráfico.

Voltando ao processo do **"Listener"**, esse é realizado através de uma **"interface"** que não significa **"interface gráfica"**, mas um outro conceito, **"interface"** dentro do JAVA são na verdade modelos que devem ser implementados em classes as **"interfaces"** definem atributos e/ou métodos cujo os códigos devem obrigatoriamente **"escritos"** pelas classes que implementam essas **"interfaces"**.

Devido a extensão do código vou detalhar apenas as partes que sofreram alterações:

```
14 public class FormMedia extends JFrame implements ActionListener{
```

Foi adicionada a classe a implementação da “interface” “**ActionListener**” isso nos obriga a implementar seu único método “**actionPerformed**”. A imagem abaixo ilustra o código do método “**actionPerformed**”:

```
71 @Override
72 public void actionPerformed(ActionEvent e) {
73
74     if(e.getSource() == btnMedia){
75
76         this.calculaMedia();
77
78     }else{
79
80         this.limparCampos();
81
82     }
83
84 }
```

Observe que na linha setenta (70) o método “**actionPerformed**” exige um parâmetro que na verdade é um objeto da classe “**ActionEvent**” que será representado dentro do método pela variável “e”, esse objeto é passado pelo conceito “**Listener**” do JAVA que é ativado automaticamente quando um componente gráfico sofre uma ação de evento, essa variável nos permite testar de qual componente gráfico veio o evento e trata-lo de acordo. Observe as linhas de setenta e quatro (74) a oitenta e dois (82) onde foi implementada uma estrutura “**if...else**” que testa se o evento partiu do botão calcular “**if**” ou se partiu do botão limpar “**else**” e chama o método específico ao tratamento de cada botão.

A imagem abaixo ilustra o código do método “**calculaMedia()**” que é a resposta ao evento de clique no botão calcular:

```
86 private void calculaMedia() {  
87  
88     double n1 = Double.parseDouble(txtN1.getText());  
89     double n2 = Double.parseDouble(txtN2.getText());  
90     double n3 = Double.parseDouble(txtN3.getText());  
91     double n4 = Double.parseDouble(txtN4.getText());  
92  
93     double media = (n1 + n2 + n3 + n4) / 4;  
94  
95     lblMedia.setText(String.valueOf(media));  
96  
97 }
```

São declaradas quatro variáveis locais do tipo “**double**” onde cada uma vai receber um valor respectivo a cada caixa de texto do formulário, observe que todo valor vindo de uma caixa de texto é considerado como “**String**” mesmo quando é composto apenas por números, isso nos obriga a converter esse dado para “**double**” através do método “**parseDouble**” da classe “**Double**” e note também como o valor é recuperado da caixa de texto através do método “**getText**” da classe “**TextField**”:

double n1 = Double.parseDouble(txtN1.getText());

A linha noventa três calcula a média e atribui o resultado a uma variável local chamada de “media” e finalmente na linha noventa e cinco (95) o valor é configurado na “**label**” de resultado: “**lblMedia.setText(String.valueOf(media))**”, observe o uso método “**valueOf**” da classe “**String**” que converte o valor da variável “**media**” para texto.

A imagem abaixo ilustra o código do método “**calculaCampos()**”:

```
99  private void limparCampos() {  
100  
101      txtN1.setText("");  
102      txtN2.setText("");  
103      txtN3.setText("");  
104      txtN4.setText("");  
105      lblMedia.setText("0.00");  
106  
107  }
```

A lógica aplicada nesse método é bastante simples, cada campo de texto recebe o valor de uma “**string**” vazia através método “**setText**” da classe “**JTextField**” com exceção de “**lblMedia**” que tem seu conteúdo configurado com “0.00”.

Por fim falta a parte mais importante que é registrar o “**ActionListener**” aos componentes que vão sofrer os eventos no caso os botões de calcular e limpar de nosso formulário.

A imagem abaixo ilustra o código do registro do “**ActionListener**” nos botões:

```
btnMedia.addActionListener(this);  
btnLimpar.addActionListener(this);
```

Nesse caso essas linhas corresponde as linhas de número sessenta (60) e sessenta e um (61) e estão dentro do método construtor, observe o uso do método “**addActionListener**” que recebe como parâmetro o valor “**this**” que indica que o método “**actionPerformed**” se encontra na mesma classe.

Validando os campos da aplicação de média.

Vamos criar um processo de validação simples que vai consistir em duas etapas, primeiro verificar se os campos não estão vazios e segundo verificar se o valor digitado é um número.

Para tanto vamos adicionar mais um método em na classe do formulário chamado “**validaCampos**” as imagens abaixo ilustram a implementação do código:

```
111 public boolean validaCampos() {
112
113     //Remove os espaços em branco a esquerda e a direita do valores
114     //informados pelo usuário.
115     String n1 = txtN1.getText().trim();
116     String n2 = txtN2.getText().trim();
117     String n3 = txtN3.getText().trim();
118     String n4 = txtN4.getText().trim();
119 }
```

A linha cento e onze (111) declara o método com um tipo de retorno booleano, isso se faz necessário pois só vamos realizar o cálculo da média se o método retornar verdadeiro no final de sua execução e isso só vai acontecer se os dados de todos os campos forem validos.

Das linhas cento e quinze (115) a cento e dezoito (118) realizamos a remoção de possíveis espaços em brancos digitados acidentalmente pelo usuário através do método “trim” da classe “**JTextField**” e atribuímos esse valor a variáveis locais.

```
120 //Testa se os campos estão vazios.
121 if (n1.equals("")) {
122
123     JOptionPane.showMessageDialog(this, "Valor não informado", "Erro", JOptionPane.ERROR_MESSAGE);
124     txtN1.requestFocus();
125     return false;
126 }
127
128 if (n2.equals("")) {
129
130     JOptionPane.showMessageDialog(this, "Valor não informado", "Erro", JOptionPane.ERROR_MESSAGE);
131     txtN2.requestFocus();
132     return false;
133 }
134
135 if (n3.equals("")) {
136
137     JOptionPane.showMessageDialog(this, "Valor não informado", "Erro", JOptionPane.ERROR_MESSAGE);
138     txtN3.requestFocus();
139     return false;
140 }
141
142 if (n4.equals("")) {
143
144     JOptionPane.showMessageDialog(this, "Valor não informado", "Erro", JOptionPane.ERROR_MESSAGE);
145     txtN4.requestFocus();
146     return false;
147 }
```

Na sequência temos uma bateria de “**ifs**” que verificam se os campos estão vazios, observe as linhas de cento vinte um (121) até cento e vinte seis (126), primeiro o “**if**” testa se a variável local “**n1**” é vazia, observe que isso é feito através do método “**equals**” da classe “**String**” não é possível usar operadores de comparação com texto no **JAVA**, caso a condição seja verdadeira o fluxo do programa será desviado para o “**corpo**” do “**if**” onde primeiro é chamado o método “**showMessageDialog**” da classe “**JOptionPane**” que vai gerar uma caixa de diálogo informando ao usuário que algo está errado, observe que são passados quatro parâmetros para esse método o primeiro faz referência ao “**JFrame**” e vai centralizar caixa de diálogo em relação a esse o segundo é a mensagem que vai ser exibida o terceiro é título da caixa de diálogo e o quarto e último é o ícone que será exibido a esquerda da mensagem.

Na linha seguinte temos “**txtN1.requestFocus();**” que vai devolver o foco do cursor para a caixa de texto onde ocorreu o erro, nesse caso a caixa do primeiro número.

E por o método retorna o valor “**false**” isso aborta a execução do método de validação e dá a oportunidade do usuário digitar um novo valor na caixa de texto.

O processo se repete para todas as caixas de texto.

```
168 //Testa se os valores informados são números.
169 try {
170     Double.parseDouble(n1);
171 } catch (Exception e) {
172     JOptionPane.showMessageDialog(this, "Valor inválido", "Erro", JOptionPane.ERROR_MESSAGE);
173     txtN1.setText("");
174     txtN1.requestFocus();
175     return false;
176 }
177
178 try {
179     Double.parseDouble(n2);
180 } catch (Exception e) {
181     JOptionPane.showMessageDialog(this, "Valor inválido", "Erro", JOptionPane.ERROR_MESSAGE);
182     txtN2.setText("");
183     txtN2.requestFocus();
184     return false;
185 }
186
187 try {
188     Double.parseDouble(n3);
189 } catch (Exception e) {
190     JOptionPane.showMessageDialog(this, "Valor inválido", "Erro", JOptionPane.ERROR_MESSAGE);
191     txtN3.setText("");
192     txtN3.requestFocus();
193     return false;
194 }
195
196 try {
197     Double.parseDouble(n4);
198 } catch (Exception e) {
199     JOptionPane.showMessageDialog(this, "Valor inválido", "Erro", JOptionPane.ERROR_MESSAGE);
200     txtN4.setText("");
201     txtN4.requestFocus();
202     return false;
203 }
204
205 }
```

O trecho de código acima verifica se o valor digitado é realmente um número, nesse caso utilizamos o método “**parseDouble**” da classe “**Double**” passando para esse como parâmetro o valor a ser testado, porém nesse caso não podemos utilizar um “**if**” pois o método não retorna verdadeiro ou falso e sim valor um numérico convertido caso seja possível se não for possível converter o valor o método devolve uma exceção (erro), então fazemos uso da estrutura “**try...catch**” do JAVA que serve justamente para a questão do tratamento de exceção, onde o comando “**try**” (tentar) vai tentar executar uma ação, nesse caso realizar a conversão de tipos (linha 152), caso logre êxito nada ocorre

pois significa que o valor em questão é um número, caso não seja possível realizar a conversão (o usuário digitou um alfanumérico) o método **"parseDouble"** devolve uma exceção que é tratada pela estrutura **"catch"** (pegar) onde emitimos uma mensagem de erro com o método **"showMessageDialog"** da classe **"JOptionPane"** (linha 156) nos mesmos moldes da validação de conteúdo vista anteriormente em seguida o valor digitado pelo usuário é removido da caixa de texto **"txtN1.setText("")"** (linha 157) e o foco do cursor volta para caixa de texto onde ocorreu o erro **"txtN1.requestFocus()"** (linha 158).

E por fim o método retorna o valor **"false"** isso aborta a execução do método de validação e dá a oportunidade do usuário digitar um novo valor na caixa de texto.