



# Interface gráfica e seus componentes

Prof. Renato Pimentel

2024/2



## Sumário



### 1 Interface gráfica e seus componentes



## A interface gráfica com o usuário

(abreviação: GUI – de *Graphical User Interface*)

fornece um **conjunto de componentes** facilitando a utilização de uma aplicação;

- **Botões, caixas de texto, painéis, barras de rolagem**, etc.

Cada componente GUI é um objeto com o qual o usuário **interage**, via mouse, teclado ou outra forma de entrada.



## Ferramentas do Java para GUI



- **AWT** (*Abstract Window Toolkit*);
- **Swing** (mais componentes, maior flexibilidade);
- **JavaFX** (recente, coloca-se como sucessor do Swing).



Sistemas desenvolvidos com AWT são dependentes da plataforma, ou seja, em plataformas diferentes as interfaces gráficas podem ser exibidas de forma diferente, pois AWT usa as primitivas gráficas de cada plataforma;

Não fornece aparência e comportamento consistentes para diversas plataformas.



## Pacote Swing I



**Objetivo:** dotar uma aplicação Java com componentes GUI padronizados;

- Mesma aparência (ou semelhante) em qualquer Sistema Operacional.

Para isso ocorrer, a maior parte dos componentes Swing são escritos, manipulados e exibidos completamente em Java.



As instruções mostram como importar o pacote principal para aplicações Swing:

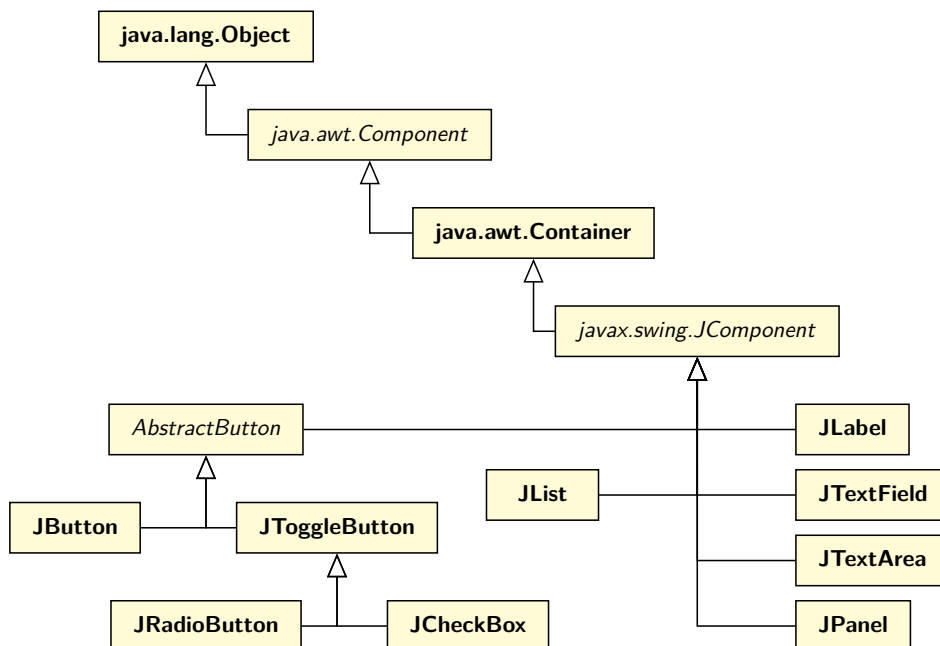
```
import javax.swing.*;  
import javax.swing.event.*;
```

A maioria das aplicações Swing também precisam de dois pacotes AWT:

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



## Pacote Swing III





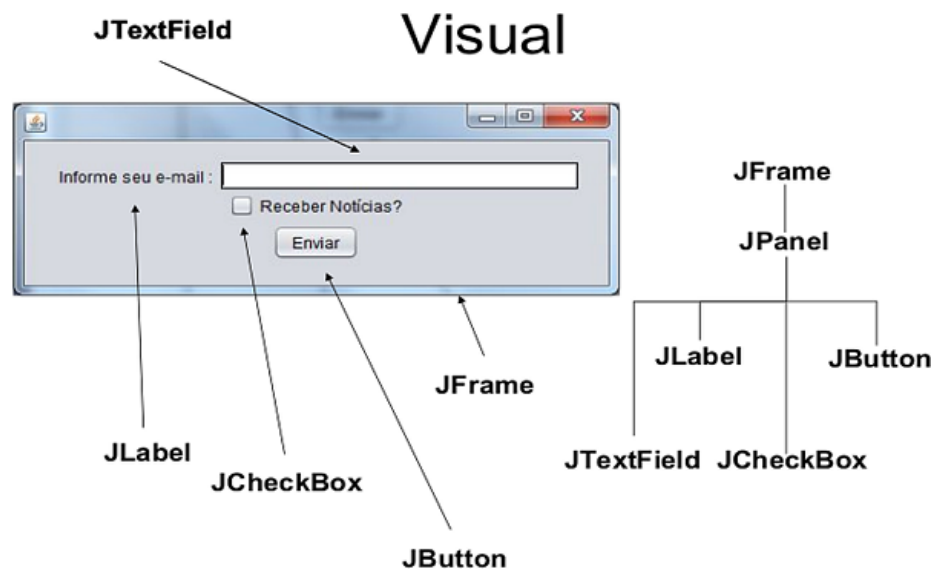
As hierarquias de herança dos pacotes `javax.swing` e `java.awt` devem ser compreendidas – especificamente a classe `Component`, a classe `Container` e a classe `JComponent`, que definem os recursos comuns à maioria dos componentes Swing:

- `Component` – define métodos que podem ser usados nas suas subclasses;
- `Container` – coleção de componentes relacionados:
  - ▶ Quando usado com `JFrames` insere componentes para o painel (um `Container`);
  - ▶ Método `add`.
- `JComponent` – superclasse da maioria dos componentes Swing.
  - ▶ Muitas das funcionalidades dos componentes são herdadas dessa classe.



Alguns componentes do Swing:

Componente	Descrição
<code>JLabel</code>	Área para exibir texto não-editável.
<code>TextField</code>	Área em que o usuário insere dados pelo teclado. Também podem exibir informações.
<code>Button</code>	Área que aciona um evento quando o mouse é pressionado.
<code>CheckBox</code>	Componentes GUI que têm dois estados: selecionado ou não.
<code>ComboBox</code>	Lista de itens a partir da qual o usuário pode fazer uma seleção clicando em um item.
<code>List</code>	Área em que uma lista de itens é exibida, a partir da qual o usuário pode fazer uma seleção clicando uma vez em qualquer elemento.
<code>Panel</code>	<i>Container</i> em que os componentes podem ser adicionados.



Para que um componente tenha certo comportamento quando ocorre um **evento** os passos abaixo devem ser seguidos:

- ① Criação de uma classe que estenda `JFrame` (define uma janela do sistema);
- ② Criação do componente visual;
- ③ Adição desse componente em um contêiner;
- ④ Criação de uma classe interna (tratador de eventos) que implemente determinado *listener*;
- ⑤ Registro do tratador de eventos ao *listener* através dos métodos disponíveis nas instâncias dos componentes GUI.

**Observação:** Para os componentes que não possuem eventos associados (ex.: `JLabel`), somente os 3 primeiros passos são aplicáveis.



Exemplo com classe JFrame:

```
1 import javax.swing.*;
2 public class Exemplo1 extends JFrame {
3     public Exemplo1() {
4         // Define o título da janela
5         super("Primeira janela");
6
7         this.setSize(320, 240); // os métodos setSize() e
8         this.setVisible(true); // setVisible são obrigatórios.
9     }
10
11     public static void main(String[] args) {
12         Exemplo1 janela = new Exemplo1();
13     }
14 }
```





Exemplo com classe JFrame:

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 public class Exemplo2 extends JFrame {
4
5     public Exemplo2() {
6         // Define o título da janela
7         super("Primeira janela");
8
9         this.setSize(320, 240);
10        this.setVisible(true);
11    }
12
13    public static void main(String[] args) {
14        Exemplo2 janela = new Exemplo2();
15    }
```



```
16 // Quando janela é fechada, apenas se torna invisível.
17 // Com comandos a seguir, será chamado o método exit(),
18 // que encerra a aplicação e libera a JVM.
19 janela.addWindowListener(
20     new WindowAdapter() { // classe do pacote awt.event
21         public void windowClosing(WindowEvent e) {
22             System.exit(0);
23         }
24     }
25 );
26 }
27 }
```





Os **gerenciadores de *layout*** organizam os componentes GUI em um contêiner para fins de apresentação.

Os principais gerenciadores de *layout* são:

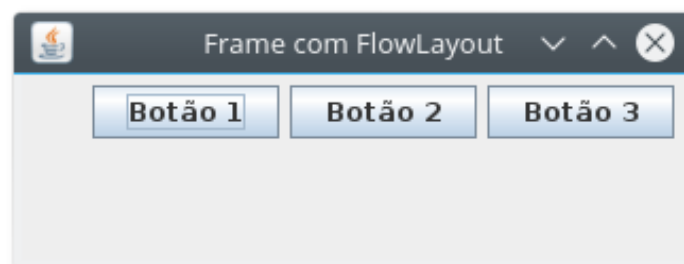
- **FlowLayout** – componentes dispostos em linha, da esquerda para a direita, na ordem em que foram adicionados.
- **BorderLayout** – componentes dispostos em 5 regiões: NORTH, SOUTH, EAST, WEST, CENTER (cada região: máximo 1).
- **GridLayout** – Área dividida em retângulos, conforme número de linhas/colunas especificados.
- **SpringLayout** – combina características dos demais; baseia-se nas relações ou restrições entre as bordas dos componentes.



```
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4
5 public class Exemplo3 extends JFrame {
6     public Exemplo3() {
7         super("Frame com FlowLayout");
8         JButton b1 = new JButton("Botão 1");
9         JButton b2 = new JButton("Botão 2");
10        JButton b3 = new JButton("Botão 3");
11        this.setSize(320, 120);
12        Container c = this.getContentPane();
13        c.add(b1);
14        c.add(b2);
15        c.add(b3);
16        c.setLayout(new FlowLayout(FlowLayout.RIGHT));
```



```
17     this.setVisible(true);  
18 }  
19  
20 // método main() aqui, como no exemplo anterior  
21 // ...  
22 }
```





- Os **rótulos** fornecem instruções de texto ou informações em um GUI;
- Os rótulos são definidos com a classe `JLabel`;
- O rótulo exibe uma **única linha de texto somente de leitura**, uma **imagem**, ou **ambos**;



### Exemplo:

Desenvolver um aplicativo Java que apresente um *label* (rótulo) no rodapé de uma tela, colocando um ícone no centro da tela, associado a uma figura de sua escolha e indicando uma *frase de dica* para o usuário, que aparecerá quando o mouse repousar sobre a figura.



```
1 package labelteste;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class LabelTeste extends JFrame {
8     private final JLabel label;
9     private final Icon icone = new ImageIcon( "java.jpg" );
10    // Configurando a GUI
11    public LabelTeste() {
12        super( "Testando JLabel" );
13        // Cria um container e define o modelo de layout (FlowLayout)
14        Container container = getContentPane();
15        container.setLayout( new FlowLayout() );
16        // JLabel sem argumentos no construtor
17        label = new JLabel();
```



```
18     label.setText("Label com ícone e texto com alinhamento de rodapé
19     bottom" );
20     label.setIcon( icone );
21     label.setHorizontalTextPosition( SwingConstants.CENTER );
22     label.setVerticalTextPosition( SwingConstants.BOTTOM );
23     label.setToolTipText("Este é o label" );
24     container.add( label );
25     setSize( 500, 300 );
26     setVisible( true );
27 }
28 // Método principal da aplicação
29 public static void main( String args[] ) {
30     LabelTeste application = new LabelTeste();
31     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
32 }
33 } // final da classe
```



**Eventos** são *mapeamentos* da **interação** do usuário com o programa.

- As GUIs são baseados em eventos, isto é, geram eventos quando o usuário interage com a interface;
- Algumas interações: *mover o mouse, clicar no mouse, clicar em um botão, digitar num campo de texto, selecionar um item de menu, fechar uma janela*, etc;
- Os eventos da GUI são enviados para o programa quando ocorre uma interação com o usuário;



- O mecanismo de tratamento de eventos possui três partes:
  - ▶ A origem do evento.
  - ▶ O objeto do evento.
  - ▶ O “ouvinte” (*listener*) do evento.
- A **origem** do evento é o componente GUI com o qual o usuário interage;
- O **objeto evento** encapsula as informações sobre o evento que ocorreu. As informações incluem uma referência para a origem do evento e quaisquer informações específicas que possam ser requeridas pelo *listener*;
- O **listener** recebe notificações de que um evento ocorreu permitindo que este realize determinada ação;



- É preciso executar as seguintes tarefas para processar um evento da GUI com o usuário em um programa:
  - ▶ registrar um *listener* para determinado componente GUI;
  - ▶ implementar um método de tratamento do evento, também chamado de **tratador de eventos** (*handler*).
- O objeto da GUI gera um `ActionEvent` (evento);
- O evento é processado por um objeto `ActionListener` (ouvinte)



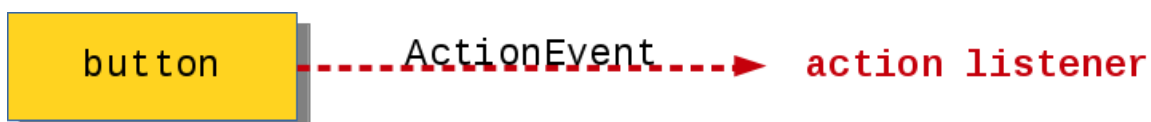
- É preciso registrar um objeto `ActionListener` na lista das ações a serem executadas. Por exemplo:

```
1 buttonCadastrar.addActionListener (  
2     new ActionListener() {  
3         public void actionPerformed(ActionEvent e) {  
4             JOptionPane.showMessageDialog(null, "Você clicou no  
5             botão!");  
6         }  
7     });
```



### Exemplo:

- Pressione um `JButton`;
- Método `actionPerformed` é chamado na escuta registrada para o objeto.

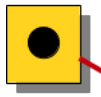




## Swing – eventos VI

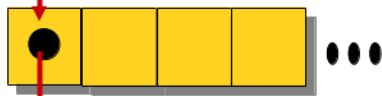


botao1



Objeto de JButton. Ele contém uma variável de instância do tipo EventListenerList chamada listenerList, que herdou da classe JComponent.

listenerList



Referência criada pela instrução  
botao1.addActionListener(gestorBotoes)

gestorBotoes



Esse objeto gestorBotoes implementa ActionListener e define o método actionPerformed().

```
public void actionPerformed(Action event){
    // evento tratado aqui
}
```

Como o tratador de eventos foi registrado?

O registro ocorre com os comandos:

FACOM32305

POO

2024/2

31 / 70

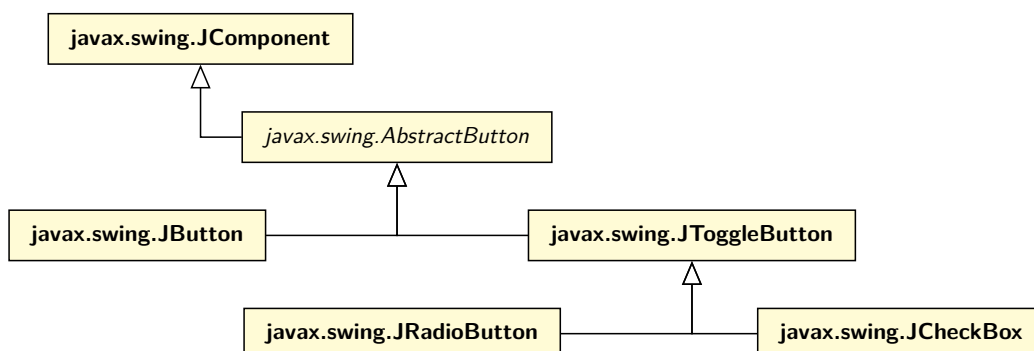
```
botao1.addActionListener( gestorBotoes );
botao2.addActionListener( gestorBotoes );
```



## Swing – JButton I



O **botão** é um componente em que o usuário clica para disparar uma ação específica. O programa Java pode utilizar vários tipos de botões, incluindo **botões de comando**, **caixas de marcação**, **botões de alternância** e **botões de opção**.



Todos são subclasses de AbstractButton (pacote javax.swing), que define muitos dos recursos comuns aos botões do Swing.





### Exemplo:

Desenvolver um aplicativo Java que apresente um botão associado a um ícone (com figura de sua escolha) e indicando uma frase de dica para o usuário (ex.: “pressione o botão”), um botão de finalização do programa e um mecanismo de tratamento do evento associado ao botão com o ícone (onde o tratamento seja apresentar uma nova janela com uma mensagem).



## Swing – JButton III



```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class ExemploButton extends JFrame {
6
7     private JButton botao1, botao2;
8     private Icon cafe = new ImageIcon("java.jpg");
9     private String strIcone = "botão associado a uma imagem";
10    private String strFinalizar = "Finalizar";
11
12    // Configura a GUI
13    public ExemploButton() {
14        super("Testando Botões");
15
16        // Cria o container e atribui o layout
17        Container container = getContentPane();
18        container.setLayout(new FlowLayout());
19    }
```



## Swing – JButton IV



```
20 // Cria os botões
21 botao1 = new JButton("Botão Java", cafe);
22 botao1.setIcon(cafe);
23
24 botao1.setToolTipText("Pressione o botão");
25 botao1.setActionCommand(strIcone);
26 container.add(botao1);
27 botao2 = new JButton(strFinalizar);
28 botao2.setToolTipText("Finaliza o programa");
29 container.add(botao2);
30 // Cria o objeto gestorBotoes (instância da classe interna
  ButtonHandler)
31 // para o uso no tratamento de eventos de botão
32 GerenciadorBotoes gestorBotoes = new GerenciadorBotoes();
33 botao1.addActionListener(gestorBotoes);
34 botao2.addActionListener(gestorBotoes);
35
36 setSize(545, 280);
37 setVisible(true);
38 }
```



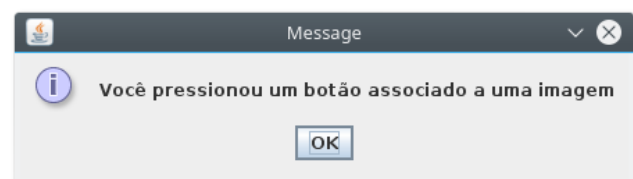
## Swing – JButton V



```
39
40 // Classe interna para tratamento de evento de botão
41 private class GerenciadorBotoes implements ActionListener {
42     // Método de manipulação do evento
43
44     public void actionPerformed(ActionEvent event) {
45         //Testa se o botão com a imagem foi pressionado
46         if (event.getActionCommand().equalsIgnoreCase(strIcone)) {
47             JOptionPane.showMessageDialog(null,
48                 "Você pressionou um " + event.getActionCommand());
49         } //Testa se o botão "Finalizar" foi pressionado
50         else if (event.getActionCommand().equalsIgnoreCase(
51             strFinalizar)) {
52             System.exit(0);
53         }
54     }
55 } // fim da classe interna GerenciadorBotoes
56
57 // Método principal
58 public static void main(String args[]) {
```



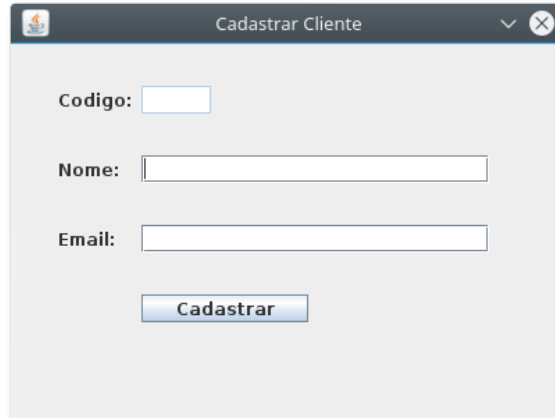
```
58     ExemploButton application = new ExemploButton();
59     application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
60 }
61
62 } // fim da classe ExemploButton
```





### Exemplo:

Desenvolver um aplicativo Java semelhante à figura abaixo:



```
1 import javax.swing.*;
2
3 public class FCliente extends JFrame {
4     private JLabel labelCodigo;
5     private JLabel labelNome;
6     private JLabel labelEmail;
7     private JTextField fieldCodigo;
8     private JTextField fieldNome;
9     private JTextField fieldEmail;
10    private JButton buttonCadastrar;
11
12    public FCliente() {
13        initComponents();
14    }
15
16    private void initComponents() {
```



## Swing – JButton X



```
17 labelCodigo = new JLabel();
18 labelNome = new JLabel();
19 labelEmail = new JLabel();
20 fieldCodigo = new JTextField();
21
22 fieldNome = new JTextField();
23 fieldEmail = new JTextField();
24 buttonCadastrar = new JButton();
25 this.setTitle("Cadastrar Cliente");
26 this.setSize(400, 300);
27 this.setResizable(false);
28 this.setDefaultCloseOperation(WindowConstants.
EXIT_ON_CLOSE);
29 this.getContentPane().setLayout(null);
30 labelCodigo.setText("Codigo:");
31 labelCodigo.setBounds(30, 30, 70, 20);
32 this.add(labelCodigo);
```



## Swing – JButton XI



```
33 labelNome.setText("Nome:");
34 labelNome.setBounds(30, 80, 70, 20);
35 this.add(labelNome);
36 labelEmail.setText("Email:");
37 labelEmail.setBounds(30, 130, 70, 20);
38 this.add(labelEmail);
39 fieldCodigo.setBounds(90, 30, 50, 20);
40 fieldCodigo.setEnabled(false);
41 this.add(fieldCodigo);
42 fieldNome.setBounds(90, 80, 250, 20);
43 this.add(fieldNome);
44 fieldEmail.setBounds(90, 130, 250, 20);
45 this.add(fieldEmail);
46 buttonCadastrar.setText("Cadastrar");
47 buttonCadastrar.setBounds(90, 180, 120, 20);
48 this.add(buttonCadastrar);
49 this.setVisible(true);
```



```
50     }
51
52     public static void main(String[] args) {
53         //Schedule a job for the event-dispatching thread:
54         //creating and showing this application's GUI.
55         javax.swing.SwingUtilities.invokeLater(new Runnable() {
56             public void run() {
57                 new FCliente();
58             }
59         });
60     }
61 } // fim da classe
```



### Exemplo:

Desenvolver um aplicativo Java que apresente quatro campos de edição, sendo um para o usuário colocar uma *frase*, outro para apresentar uma *frase editável*, outro para apresentar um *texto não-editável* e um último para *registrar senhas*. Trate os eventos associados ao acionamento da tecla Enter em cada um desses campos de edição.



```
1 // Exemplo de JTextField
2 import java.awt.*;
3 import java.awt.event.*;
4 import javax.swing.*;
5
6 public class ExemploJTextField extends JFrame {
7     private JTextField campoTexto1, campoTexto2, campoTexto3;
8     private JPasswordField campoSenha;
9
10    // configuração da GUI
11    public ExemploJTextField() {
12        super("Testando JTextField e JPasswordField");
13        Container container = getContentPane();
14        container.setLayout(new FlowLayout());
15
16        // constrói o 1o campo de texto com dimensões default
17        campoTexto1 = new JTextField(10);
18        container.add(campoTexto1);
19
20        // constrói o 2o campo de texto com texto default
```



```
21 campoTexto2 = new JTextField("Digite seu texto aqui:");
22 container.add(campoTexto2);
23
24 // constrói o 3o campo de texto com texto default e
25 // 20 elementos visíveis, sem tratador de eventos
26 campoTexto3 = new JTextField("Campo de texto não editável", 20);
27 campoTexto3.setEditable(false);
28 container.add(campoTexto3);
29
30 // constrói o 4o campo de texto com texto default
31 campoSenha = new JPasswordField("Texto oculto");
32 container.add(campoSenha);
33 // registra os tratadores de evento
34 GerenciadorTextField gerenteTexto = new GerenciadorTextField();
35 campoTexto1.addActionListener(gerenteTexto);
36 campoTexto2.addActionListener(gerenteTexto);
37 campoTexto3.addActionListener(gerenteTexto);
38 campoSenha.addActionListener(gerenteTexto);
39 setSize(360, 120);
40 setVisible(true);
```

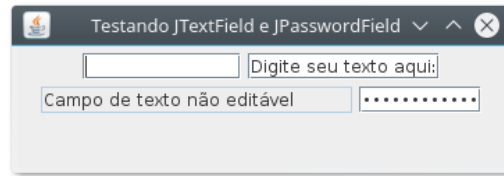


```
41     }
42
43     public static void main(String args[]) {
44         ExemploJTextField programaTexto = new ExemploJTextField();
45         programaTexto.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
46     }
47
48     //classe interna privativa para tratamento de eventos
49     private class GerenciadorTextField implements ActionListener {
50         //processa eventos de campos de texto
51         public void actionPerformed(ActionEvent evento) {
52             String texto = "";
53             // evento:Usuário pressiona ENTER no objeto de JTextField
54             campoTexto1
55             if (evento.getSource() == campoTexto1) {
56                 texto = "campoTexto1: " + evento.getActionCommand();
57             } // evento:Usuário pressiona ENTER no objeto de JTextField
58             campoTexto2
59             else if (evento.getSource() == campoTexto2) {
60                 texto = "campoTexto2: " + evento.getActionCommand();
61             }
```



```
59         } // evento:Usuário pressiona ENTER no objeto de JTextField
60         campoTexto3
61         else if (evento.getSource() == campoTexto3) {
62             texto = "campoTexto3: " + evento.getActionCommand();
63         } // evento:Usuário pressiona ENTER no objeto de
64         JPasswordField campoSenha
65         else if (evento.getSource() == campoSenha) {
66             JPasswordField senha = (JPasswordField) evento.getSource();
67             texto = "campoSenha: " + new String(campoSenha.getPassword
68             ());
69         }
70         JOptionPane.showMessageDialog(null, texto);
71     }
72 } // fim da classe interna privativa GerenciadorTextField
73 } // fim da classe ExemploJTextField
```





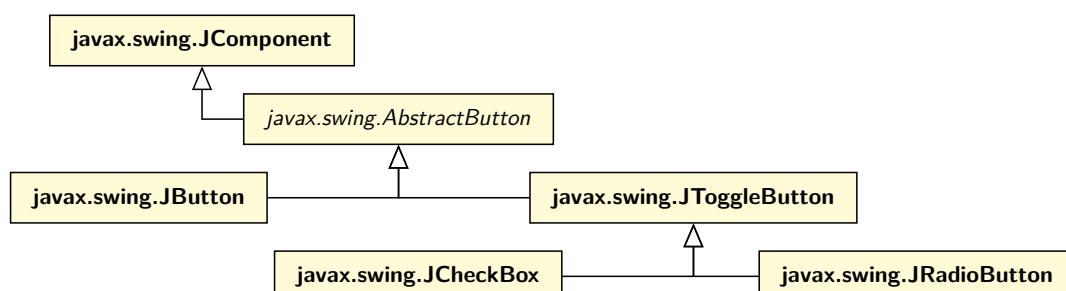
## Swing – JCheckBox e JRadioButton I



Para que o usuário interaja com um aplicativo Java, existem diversos tipos de botões para cada situação de interface.

Os componentes GUI Swing possuem três tipos de botões de estado (que assumem valores ativados/desativados ou verdadeiro/falso):

- `JToggleButton` – para barras de ferramentas;
- `JCheckBox` – para interfaces de múltipla escolha;
- `JRadioButton` – escolha única entre múltiplas alternativas.





### Exemplo:

Aplicativo Java que permita que o usuário digite uma frase e veja sua sentença aparecer em **negrito**, *itálico* ou em ***ambos***, dependendo de sua escolha.



```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class ExemploCheckBoxRadio extends JFrame {
6
7     private JCheckBox checkB, checkI;
8     private JRadioButton rbotao1, rbotao2, rbotao3;
9     private ButtonGroup grupoRadio;
10    private JPanel painel1, painel2;
11
12    // Configura a GUI
13    public ExemploCheckBoxRadio() {
14        super("Testando CheckBox e RadioButton");
15
16        // Cria o container e atribui o layout
17        Container container = getContentPane();
18        container.setLayout(new FlowLayout());
```



```
19 // Cria os painéis
20 painel1 = new JPanel();
21 painel2 = new JPanel();
22
23 // Cria os objetos CheckBox, adiciona para o painel e
24 // adiciona o painel para o container
25 checkB = new JCheckBox("Bold");
26 painel1.add(checkB);
27 checkI = new JCheckBox("Itálico");
28 painel1.add(checkI);
29 container.add(painel1);
30
31 // Cria os objetos RadioButton, adiciona para o painel e
32 //adiciona o painel para o container
33 rbotao1 = new JRadioButton("Plain", true);
34 painel2.add(rbotao1);
35 rbotao2 = new JRadioButton("Bold", false);
36 painel2.add(rbotao2);
37 rbotao3 = new JRadioButton("Itálico", false);
38 painel2.add(rbotao3);
```



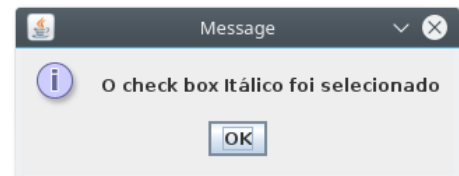
```
39 container.add(painel2);
40
41 //Cria o relacionamento lógico entre os objetos JRadioButton
42 grupoRadio = new ButtonGroup();
43 grupoRadio.add(rbotao1);
44 grupoRadio.add(rbotao2);
45
46 grupoRadio.add(rbotao3);
47
48 //Registra os tratadores de evento
49 Gerenciador gerente = new Gerenciador();
50 checkB.addItemListener(gerente);
51 checkI.addItemListener(gerente);
52 rbotao1.addItemListener(gerente);
53 rbotao2.addItemListener(gerente);
54 rbotao3.addItemListener(gerente);
55
56 setSize(300, 100);
57 setVisible(true);
58 }
```



```
59
60 // Classe interna para tratamento de evento
61 private class Gerenciador implements ItemListener {
62     // Método de manipulação do evento
63
64     public void itemStateChanged(ItemEvent event) {
65         //Testa qual objeto foi pressionado
66         if (event.getSource() == checkB) {
67
68
69             JOptionPane.showMessageDialog(null, "O check box Bold foi
selecionado");
70         } else if (event.getSource() == checkI) {
71             JOptionPane.showMessageDialog(null, "O check box Itálico
foi selecionado");
72         } else if ((event.getSource() == rbotao1)
73             && (event.getStateChange() == ItemEvent.SELECTED)) {
74             JOptionPane.showMessageDialog(null, "O radio button
Plain foi selecionado");
75         } else if ((event.getSource() == rbotao2)
```



```
76         && (event.getStateChange() == ItemEvent.SELECTED)) {
77             JOptionPane.showMessageDialog(null, "O radio button bold
foi selecionado");
78         } else if ((event.getSource() == rbotao3)
79             && (event.getStateChange() == ItemEvent.SELECTED)) {
80             JOptionPane.showMessageDialog(null, "O radio button Itá
lico foi selecionado");
81         }
82     }
83 } // fim da classe interna Gerenciador
84
85
86 // Método principal
87 public static void main(String args[]) {
88     ExemploCheckBoxRadio application = new ExemploCheckBoxRadio();
89     application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
90 }
91 } // fim da classe ExemploCheckBoxRadio
```



## Swing – menus I



Os **menus** também são parte integrante das GUIs, permitindo que a interface fique mais organizada:

- As classes utilizadas para definir menus são JMenuBar, JMenuItem, JCheckBoxMenuItem e JRadioButtonMenuItem.
- A classe JMenuBar contém os métodos necessários para gerenciar uma barra de menus;
- A classe JMenu, por sua vez, contém os métodos necessários para gerenciar menus;
- Os menus contêm itens e são adicionados à barra de menus;
- A classe JMenuItem contém os métodos necessários para gerenciar os itens dos menus;
- O item de menu é um componente GUI pertencente ao componente menu que quando selecionado realiza determinada ação;



- A classe `JCheckBoxMenuItem` contém os métodos necessários para gerenciar itens de menu que podem ser ativados ou desativados;
- A classe `JRadioButtonMenuItem` contém os métodos necessários para gerenciar itens de menu que também podem ser ativados ou desativados.



### Exemplo:

Desenvolver um aplicativo Java que apresente três menus: **Cadastro**, **Relatórios** e **Ajuda**, na barra superior da janela. O primeiro menu deve possibilitar o cadastro de Paciente e Médicos, e permitir que o sistema seja finalizado. O terceiro menu deve ter um item que possibilite a visualização de uma tela com informações do sistema (**Sobre**).



```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class ExemploMenu extends JFrame {
6
7     private JMenuBar barraMenu;
8     private JMenu mCad, mRel, mAjuda;
9     private JMenuItem iPac, iMed, iFim, iSobre;
10    private String sistema = "Sistema de Gerenciamento de Clínicas";
11    private String versao = "Versao 1.0";
12    private String build = "(build 20030626)";
13
14    // Configura a GUI
15    public ExemploMenu() {
16        //Atribui o título para a janela
17        setTitle(sistema);
18
19        //Cria a barra de menus
```



```
20    barraMenu = new JMenuBar();
21
22
23    //Cria os menus e adiciona para a barra
24    mCad = new JMenu("Cadastro");
25    mCad.setMnemonic('C');
26    mRel = new JMenu("Relatórios");
27    mRel.setMnemonic('R');
28    mAjuda = new JMenu("Ajuda");
29    mAjuda.setMnemonic('A');
30    barraMenu.add(mCad);
31    barraMenu.add(mRel);
32    barraMenu.add(mAjuda);
33    //Cria os itens de menu
34    iPac = new JMenuItem("Paciente");
35    iPac.setMnemonic('P');
36    iMed = new JMenuItem("Médico");
37    iMed.setMnemonic('M');
38    iFim = new JMenuItem("Finaliza");
39    iFim.setMnemonic('F');
```



## Swing – menus VI



```
40     iSobre = new JMenuItem("Sobre");
41     iSobre.setMnemonic('S');
42     //Adiciona os itens para o menu de Cadastro
43     mCad.add(iPac);
44     mCad.add(iMed);
45     mCad.addSeparator();
46
47     mCad.add(iFim);
48     //Adiciona o item sobre para o menu Ajuda
49     mAjuda.add(iSobre);
50
51     // registra os tratadores de evento
52     Gerenciador gerente = new Gerenciador();
53     iPac.addActionListener(gerente);
54     iFim.addActionListener(gerente);
55     iSobre.addActionListener(gerente);
56
57     //Anexa a barra de menu a janela
58     setJMenuBar(barraMenu);
59     setSize(800, 600);
```



## Swing – menus VII



```
60     setVisible(true);
61     //Configura para permitir o fechamento da aplicação
62     //quando a janela for fechada
63     setDefaultCloseOperation(EXIT_ON_CLOSE);
64 }
65
66 private class Gerenciador implements ActionListener {
67     //processa eventos de campos de texto
68
69
70     public void actionPerformed(ActionEvent evento) {
71         if (evento.getSource() == iPac) {
72             //ExemploGridBagLayout cadastro = new ExemploGridBagLayout
73             ();
74         } else if (evento.getSource() == iFim) {
75             System.exit(0);
76         } else if (evento.getSource() == iSobre) {
77             JOptionPane.showMessageDialog(null,
78                 sistema + "\n\n" + " " + versao + " " + build +
79                 "\n\n",
```





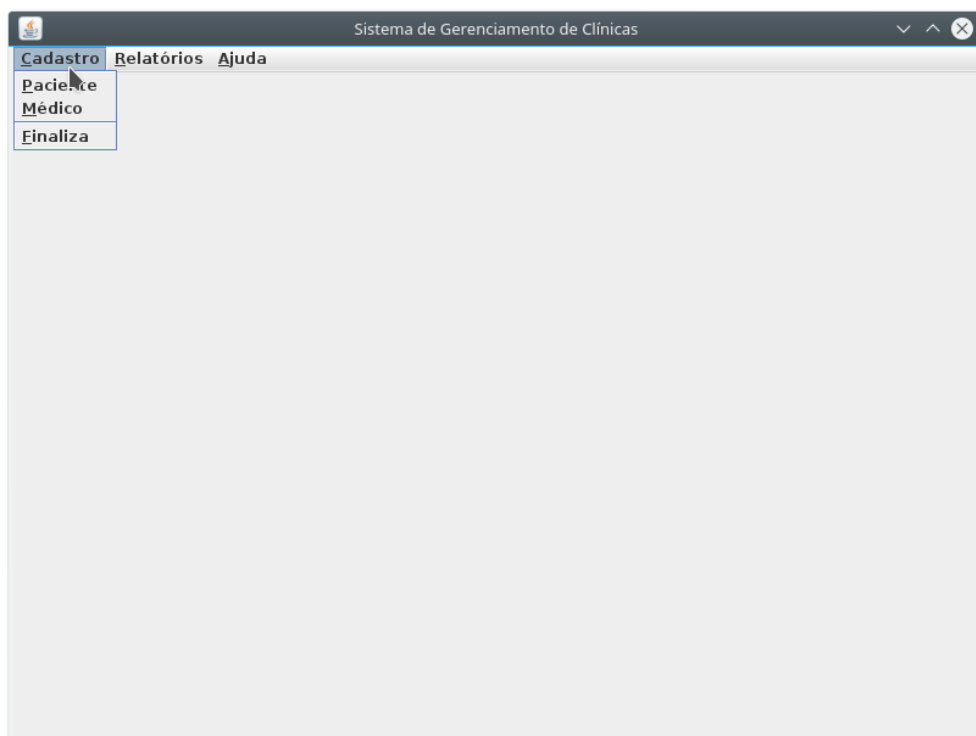
## Swing – menus VIII



```
78         "Sobre o sistema", JOptionPane.PLAIN_MESSAGE);
79     }
80 }
81 }
82
83 public static void main(String arg[]) {
84     ExemploMenu menuGeral = new ExemploMenu();
85 }
86
87 } //Fim da classe ExemploMenu
```



## Swing – menus IX

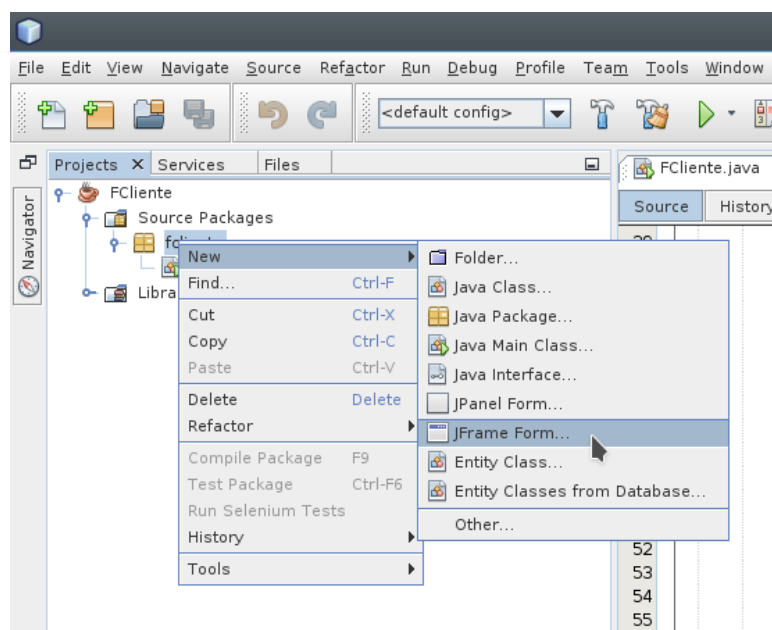


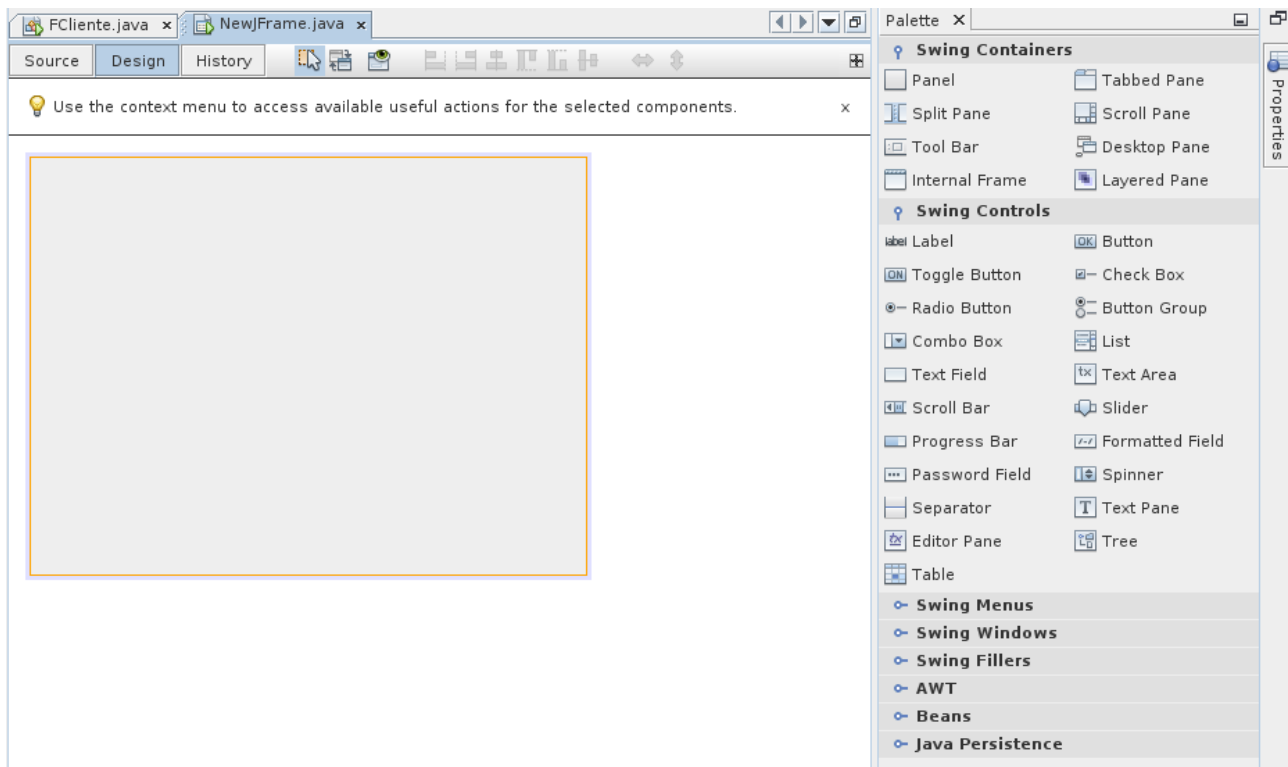


Como visto no código anterior, o método `setMnemonic(char)` permite definir o caractere de atalho utilizado conjuntamente com a tecla Alt.



## No NetBeans





## Referências e links úteis



Os slides de parte desta seção foram cedidos por Marcelo Z. do Nascimento, FACOM/UFU

LaTeXagem e adaptações: Renato Pimentel, FACOM/UFU

Veja também:

- <https://docs.oracle.com/javase/tutorial/uiswing/start/index.html>
- [https://netbeans.org/kb/docs/java/quickstart-gui\\_pt\\_BR.html#project](https://netbeans.org/kb/docs/java/quickstart-gui_pt_BR.html#project) (GUI NetBeans)
- [https://netbeans.org/kb/docs/java/gui-image-display\\_pt\\_BR.html](https://netbeans.org/kb/docs/java/gui-image-display_pt_BR.html) (tratando imagens GUI NetBeans)
- <http://slideplayer.com.br/slide/10378301> (eventos em Swing)