

## Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

*FACOM32305 - Programação Orientada a Objetos*

Prof. Thiago Pirola Ribeiro

## 1 Interface gráfica e seus componentes

A **interface gráfica com o usuário** (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.

- **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.

**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.

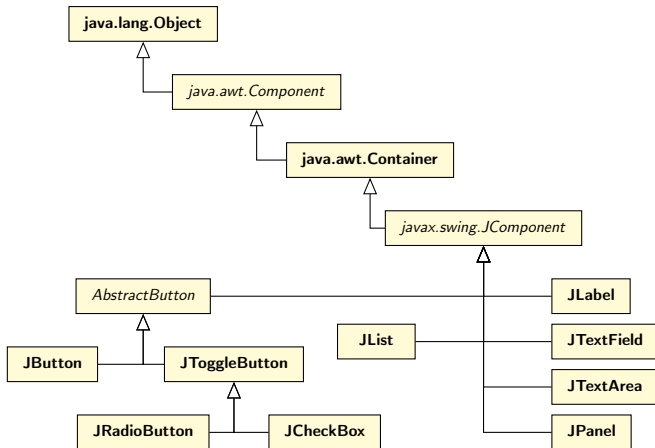
As instruções mostram como importar o pacote principal para aplicações Swing:

```
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



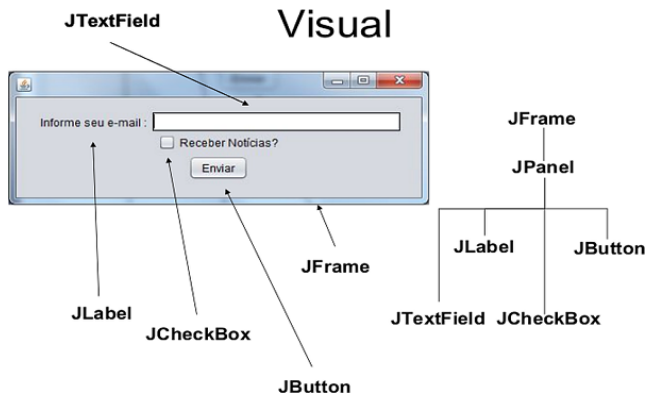


As hierarquias de herança dos pacotes `javax.swing` e `java.awt` devem ser compreendidas – especificamente a classe `Component`, a classe `Container` e a classe `JComponent`, que definem os recursos comuns à maioria dos componentes Swing:

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

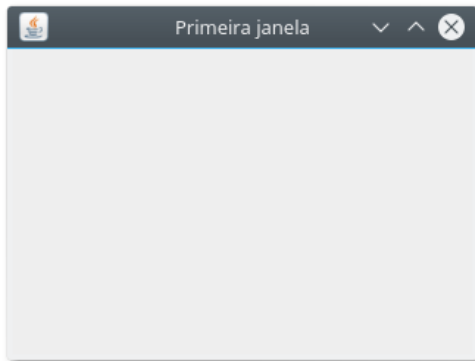


# Swing – exemplos básicos I

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 }
```

# Swing – exemplos básicos II



# Swing – exemplos básicos III

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         this.setSize(320, 240);
9         this.setVisible(true);
10    }
11
12    public static void main(String[] args) {
13        Exemplo2 janela = new Exemplo2();
14
15        // Quando janela é fechada, apenas se torna invisível.
```

# Swing – exemplos básicos IV

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

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.



# Swing – *layout* II

```
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.setLayout(new FlowLayout(FlowLayout.RIGHT));
14        c.add(b1);
15        c.add(b2);
```

# Swing – *layout* III

```
16     c.add(b3);
17     this.setVisible(true);
18 }
19
20 // método main() aqui, como no exemplo anterior
21 // ...
22 }
```

# Swing – *layout* IV



## Exercícios

Pesquise sobre o BorderLayout e o GridLayout e implemente a janela do exemplo anterior utilizando estes dois Layouts.

- 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**;

# Swing – rótulos e botões II

```
1 package labelteste;
2 import java.awt.*;
3 import java.awt.event.*;
4 import javax.swing.*;
5
6 public class LabelTeste extends JFrame {
7     private final JLabel label;
8     // Configurando a GUI
9     public LabelTeste() {
10         super( "Testando JLabel" );
11         JButton b1 = new JButton("Botão 1");
12         JButton b2 = new JButton("Botão 2");
13         JButton b3 = new JButton("Botão 3");
14         // Cria um container e define o modelo de layout
15         Container container = getContentPane();
16         container.setLayout( new GridLayout(0, 2, 30, 30));
17         // JLabel sem argumentos no construtor
```

# Swing – rótulos e botões III

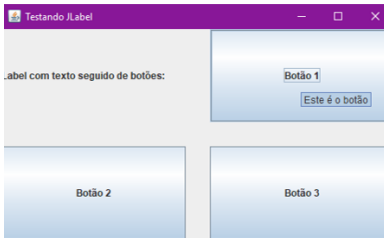
```
18     label = new JLabel();
19     label.setText("Label com texto seguido de botões:" );
20     label.setHorizontalTextPosition( SwingConstants.CENTER );
21     label.setVerticalTextPosition( SwingConstants.BOTTOM );
22     label.setToolTipText("Este é o label" );
23     b1.setToolTipText("Este é o botão");
24     container.add( label );
25     container.add(b1);
26     container.add(b2);
27     container.add(b3);
28     setSize( 500, 300 );
29     setVisible( true );
30 }
31
32 // Método principal da aplicação
33 public static void main( String args[] ) {
34     LabelTeste application = new LabelTeste();
35     application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
36 }
```

# Swing – rótulos e botões IV

```
37 } // final da classe
```



# Swing – rótulos e botões V



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

- As GUIs são baseadas 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;

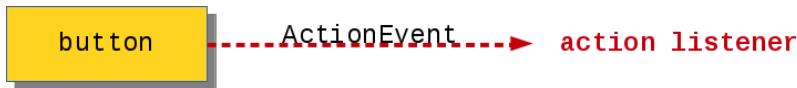
- É preciso executar as seguintes tarefas para processar um evento da GUI com o usuário em um programa:
  - registrar um *listener* para determinado componente GUI;
  - implementar um método de tratamento do evento, também chamado de **tratador de eventos** (*handler*).
- O objeto da GUI gera um `ActionEvent`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

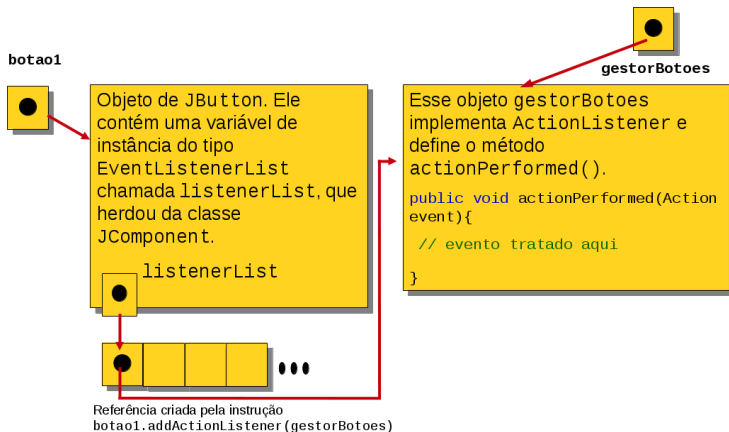
```
1 buttonCadastrar.addActionListener (
2     new ActionListener() {
3         public void actionPerformed(ActionEvent e) {
4             JOptionPane.showMessageDialog(null, "Você clicou no
5             botão!");
6         }
7     });
```

## Exemplo:

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



# Swing – eventos VI



Como o tratador de eventos foi registrado?

O registro ocorre com os comandos:

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

Eles adicionaram o *handler* de Button (gestorBotoes) como sendo um *listener* para os objetos botao1 e botao2.

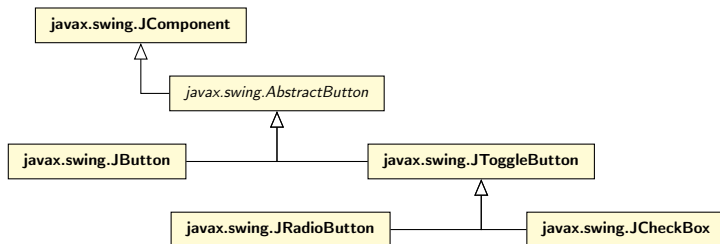


Como o componente “sabe” que deve chamar `actionPerformed()`, em vez de outro método de tratamento de eventos?

- Cada `JComponent` suporta vários tipos de eventos (de mouse, de teclado, etc.).
- Quando ocorre um evento, é acionado para os ouvintes de eventos do tipo apropriado. O despacho do evento é simplesmente uma chamada ao método de tratamento de eventos para cada ouvinte registrado para esse tipo de evento.
- Cada tipo de evento tem uma interface *listener* de eventos correspondente.
- Por exemplo, `ActionEvents` são tratados por `ActionListeners`; `MouseEvent`s por `MouseListeners` (e `MouseMotionListeners`) e `KeyEvent`s por `KeyListeners`.

# Swing – Botões e Caixas de Texto 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.

# Swing – Botões e Caixas de Texto II

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class ExemploJButton1 extends JFrame {
6
7     private JButton botao1, botao2;
8     private String strMsg = "botão do exemplo 1";
9     private String strFinalizar = "Finalizar";
10
11     // Configura a GUI
12     public ExemploJButton1() {
13         super("Testando Botões");
14         // Cria o container e atribui o layout
15         Container container = getContentPane();
16         container.setLayout(new FlowLayout());
17         // Cria os botões
18         botao1 = new JButton("Botão exemplo");
```

# Swing – Botões e Caixas de Texto III

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

# Swing – Botões e Caixas de Texto IV

```
37 private class GerenciadorBotoes implements ActionListener {
38     // Método de manipulação do evento
39
40     public void actionPerformed(ActionEvent event) {
41         //Testa se o botão exemplo foi pressionado
42         if (event.getActionCommand().equalsIgnoreCase(strMsg)) {
43             JOptionPane.showMessageDialog(null,
44                 "Você pressionou um " + event.getActionCommand());
45         } //Testa se o botão "Finalizar" foi pressionado
46         else if (event.getActionCommand().equalsIgnoreCase(
47             strFinalizar)) {
48             System.exit(0);
49         }
50     } // fim da classe interna GerenciadorBotoes
51
52     // Método principal
53     public static void main(String args[]) {
54         ExemploJButton1 application = new ExemploJButton1();
```

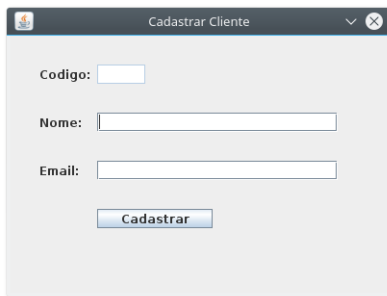
# Swing – Botões e Caixas de Texto V

```
55     application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
56 }  
57  
58 } // fim da classe ExemploJButton1
```

# Swing – Botões e Caixas de Texto VI

## Exemplo:

Desenvolver um aplicativo Java semelhante à figura abaixo:



The image shows a Java Swing window titled "Cadastrar Cliente". The window has a standard title bar with a minimize button, a maximize button, and a close button. Inside the window, there are three text input fields arranged vertically. The first field is labeled "Codigo:", the second is labeled "Nome:", and the third is labeled "Email:". Below these fields is a button labeled "Cadastrar". The window has a light gray background and a dark gray title bar.

# Swing – Botões e Caixas de Texto VII

```
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
```



# Swing – Botões e Caixas de Texto VIII

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

# Swing – Botões e Caixas de Texto IX

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

# Swing – Botões e Caixas de Texto X

```
49     }  
50  
51     public static void main(String[] args) {  
52         new FCliente();  
53     }  
54 } // 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.

# Swing – Botões e Caixas de Texto XII

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

# Swing – Botões e Caixas de Texto XIII

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

# Swing – Botões e Caixas de Texto XIV

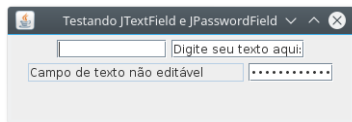
```
37     setSize(360, 120);
38     setVisible(true);
39 }
40
41 public static void main(String args[]) {
42     ExemploJTextField programaTexto = new ExemploJTextField();
43     programaTexto.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44 }
45
46 //classe interna privativa para tratamento de eventos
47 private class GerenciadorTextField implements ActionListener {
48     //processa eventos de campos de texto
49     public void actionPerformed(ActionEvent evento) {
50         String texto = "";
51         // evento:Usuário pressiona ENTER no objeto de JTextField
52         campoTexto1
53         if (evento.getSource() == campoTexto1) {
54             texto = "campoTexto1: " + evento.getActionCommand();
```

# Swing – Botões e Caixas de Texto XV

```
54     } // evento:Usuário pressiona ENTER no objeto de
JTextField campoTexto2
55     else if (evento.getSource() == campoTexto2) {
56         texto = "campoTexto2: " + evento.getActionCommand();
57     } // evento:Usuário pressiona ENTER no objeto de
JTextField campoTexto3
58     else if (evento.getSource() == campoTexto3) {
59         texto = "campoTexto3: " + evento.getActionCommand();
60     } // evento:Usuário pressiona ENTER no objeto de
JPasswordField campoSenha
61     else if (evento.getSource() == campoSenha) {
62         texto = "campoSenha: " + new String(campoSenha.
getPassword());
63     }
64     JOptionPane.showMessageDialog(null, texto);
65 }
66 } // fim da classe interna privativa GerenciadorTextField
67 } // fim da classe ExemploJTextField
```



# Swing – Botões e Caixas de Texto XVI

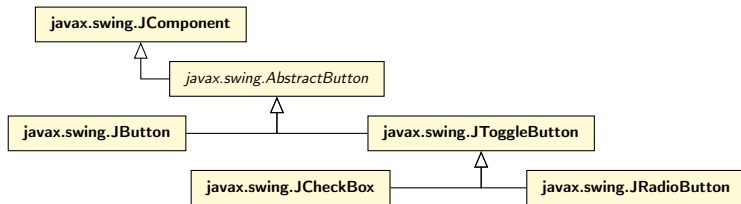


# Swing – JCheckBox e JRadioButton I

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.

# Swing – JCheckBox e JRadioButton III

```
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();
```

# Swing – JCheckBox e JRadioButton IV

```
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);
```

# Swing – JCheckBox e JRadioButton V

```
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
```

# Swing – JCheckBox e JRadioButton VI

```
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
70 foi selecionado");
71         } else if (event.getSource() == checkI) {
72             JOptionPane.showMessageDialog(null, "O check box Itá
73 lico foi selecionado");
74         } else if ((event.getSource() == rbotao1)
```

# Swing – JCheckBox e JRadioButton VII

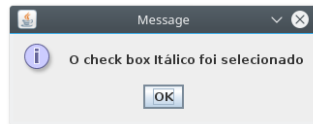
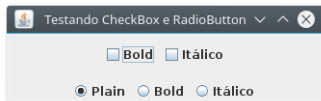
```
73         && (event.getStateChange() == ItemEvent.SELECTED)) {
74             JOptionPane.showMessageDialog(null, "0 radio button
Plain foi selecionado");
75         } else if ((event.getSource() == rbotao2)
76             && (event.getStateChange() == ItemEvent.SELECTED)) {
77             JOptionPane.showMessageDialog(null, "0 radio button
bold foi selecionado");
78         } else if ((event.getSource() == rbotao3)
79             && (event.getStateChange() == ItemEvent.SELECTED)) {
80             JOptionPane.showMessageDialog(null, "0 radio button
Itálico foi selecionado");
81         }
82     }
83 } // fim da classe interna Gerenciador
84
85
86 // Método principal
87 public static void main(String args[]) {
```



# Swing – JCheckBox e JRadioButton VIII

```
88     ExemploCheckBoxRadio application = new ExemploCheckBoxRadio()  
89     ;  
89     application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
90 }  
91 } // fim da classe ExemploCheckBoxRadio
```

# Swing – JCheckBox e JRadioButton IX



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 `JItemMenu` 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 um tela com informações do sistema (**Sobre**).

# Swing – menus IV

```
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, mAjudas;
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    }
```

# Swing – menus V

```
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 mAjudas = new JMenu("Ajuda");
29 mAjudas.setMnemonic('A');
30 barraMenu.add(mCad);
31 barraMenu.add(mRel);
32 barraMenu.add(mAjudas);
33 //Cria os itens de menu
34 iPac = new JMenuItem("Paciente");
35 iPac.setMnemonic('P');
36 iMed = new JMenuItem("Médico");
37 iMed.setMnemonic('M');
```

# Swing – menus VI

```
38 iFim = new JMenuItem("Finaliza");
39 iFim.setMnemonic('F');
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
```



# Swing – menus VII

```
57 //Anexa a barra de menu a janela
58 setJMenuBar(barraMenu);
59 setSize(800, 600);
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         } else if (evento.getSource() == iFim) {
74             System.exit(0);
```

# Swing – menus VIII

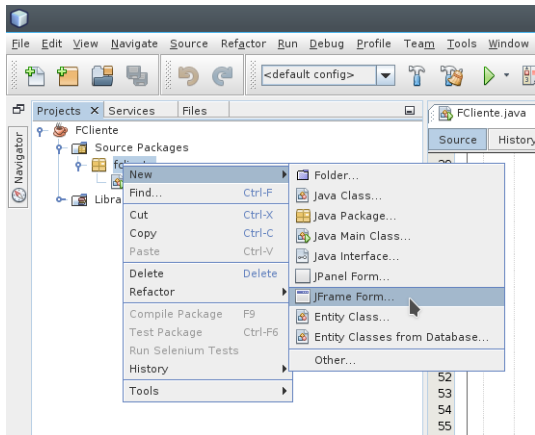
```
75         } else if (evento.getSource() == iSobre) {
76             JOptionPane.showMessageDialog(null,
77                 sistema + "\n\n" + " " + versao + " " +
78                 build + "\n\n",
79                 "Sobre o sistema", JOptionPane.PLAIN_MESSAGE);
80         }
81     }
82
83     public static void main(String arg[]) {
84         ExemploMenu menuGeral = new ExemploMenu();
85     }
86
87 } //Fim da classe ExemploMenu
```

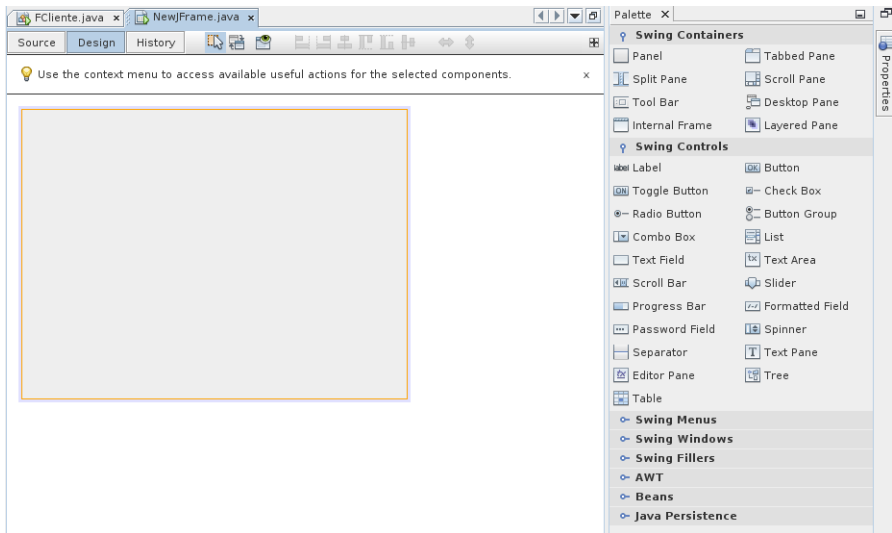
# Swing – menus IX



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.apache.org/tutorial/main/kb/docs/java/quickstart-gui/>  
(GUI NetBeans)
- <https://netbeans.apache.org/tutorial/main/kb/docs/java/gui-image-display/>  
(tratando imagens GUI NetBeans)
- <http://slideplayer.com.br/slide/10378301> (eventos em Swing)