

<b>1</b>	<b>Interfaces Gráficas com Java</b>	<b>4</b>
1.1	AWT (Abstract Window Toolkit)	4
1.1.1	Janelas	5
1.1.2	Criando uma janela simples	6
1.1.3	Adicionando um componente ao container	7
1.2	Swing	8
1.2.1	Containers	8
1.2.2	Criando uma janela simples	9
1.2.3	Adicionando um componente ao container	10
1.2.4	Laboratório 1	11
1.3	Principais Componentes Swing	12
1.3.1	javax.swing.JButton	12
1.3.2	javax.swing.JTextField	14
1.3.3	javax.swing.JComboBox	15
1.4	Gerenciadores de Layout	17
1.4.1	java.awt.FlowLayout	18
1.4.2	java.awt.GridLayout	19
1.4.3	java.awt.BorderLayout	21
1.4.4	Laboratório 2	23
1.5	Eventos	24
1.5.1	Listeners	25
1.5.2	Uma classe externa implementa a interface java.awt.XXXListener	27
1.5.3	Laboratório 3	29
1.5.4	A própria classe implementa a interface java.awt.event.XXXListener	30
1.5.5	Implementando o listener como uma classe interna (Inner class)	32
1.5.6	Laboratório 4	34
1.5.7	Implementando o listener como uma classe anônima	35
1.5.8	A interface java.awt.event.ActionListener	37
1.5.9	Laboratório 5	40
<b>2</b>	<b>Input &amp; Output</b>	<b>41</b>
2.1	Stream binários	43
2.1.1	InputStream & OutputStream	43
2.1.2	DataInputStream & DataOutputStream	57
2.1.3	BufferedInputStream & BufferedOutputStream	61
2.1.4	java.io.PrintStream	69
2.1.5	ObjectInputStream & ObjectOutputStream	70
2.1.6	Laboratório 6	76
2.2	File	77
2.2.2	FilenameFilter e FileFilter	83
2.2.3	Laboratório 7	86
2.3	Streams de caracteres	87
2.3.1	Reader & Writer	87
2.3.2	FileReader & FileWriter	90
2.3.3	BufferedReader & BufferedWriter	91
2.3.4	InputStreamReader & OutputStreamWriter	93
2.3.5	PrintWriter	95
2.3.6	Laboratório 8	99

<b>3</b>	<b>Multi-Threads</b>	<b>100</b>
3.1	Threads	100
3.2	JVM & Threads	100
3.3	Overview das classes	101
3.3.1	java.lang.Runnable	101
3.3.2	java.lang.Thread	101
3.3.3	Criando novas Threads estendendo a classe Thread	102
3.3.4	Criando novas Threads implementando a interface Runnable	104
3.4	Laboratório 8	108
3.5	Estados das Threads	109
3.6	Sincronismo	110
3.6.1	synchronized	115
3.7	wait e notify / notifyAll	118
3.7	Laboratório 9	124
3.8	Laboratório 10	125
<b>4</b>	<b>Apêndice A : Componentes Avançados do Swing</b>	<b>126</b>
4.1	javax.swing.JCheckbox	126
4.2	javax.swing.JList	128
4.3	Construindo Menus	130
4.4	A classe javax.swing.JFileChooser	132
4.5	javax.swing.JTabbedPane	134
4.6	javax.swing.JTree	135
4.7	java.awt.CardLayout	137
<b>5</b>	<b>Apêndice B: Applets</b>	<b>140</b>
5.1	Ciclo de vida de um Applet	141
5.2	Visualizando um applet	142
5.3	Principais métodos	143
5.4	Métodos especiais para Web	145
5.5	Passando parâmetros para um Applet	147
<b>6</b>	<b>Apêndice C : Networking</b>	<b>149</b>
6.1	Socket	150
6.2	ServerSocket	154
6.3	Exemplos Adicionais	156
<b>7</b>	<b>Apêndice D: Reflection API</b>	<b>160</b>
7.1	java.lang.Class	161
7.2	java.lang.reflect.Field	165
7.3	java.lang.reflect.Constructor	167
7.4	java.lang.reflect.Method	168
7.5	java.lang.reflect.Modifier	169
7.6	Montando uma Central de Testes para as classes de Reflection	170
<b>8</b>	<b>Apêndice E: JavaMail</b>	<b>172</b>
8.1	javax.mail.Session	172
8.2	javax.mail.internet.MimeMessage	173
8.3	javax.mail.internet.InternetAddress	173
8.4	javax.mail.Transport	174
8.5	javax.mail.Authenticator	174

8.6	javax.mail.PasswordAuthentication	174
8.7	classe javax.mail.Store	174
8.8	classe javax.mail.Folder	175
8.9	Envio de Mensagens	175
8.10	Leitura de Mensagens	176
8.11	Exemplos	177

**Globalcode®**  
THE DEVELOPERS COMPANY

**Modelo de Apresentação**

Atenção: A Cópia integral ou parcial  
deste material não está autorizada.

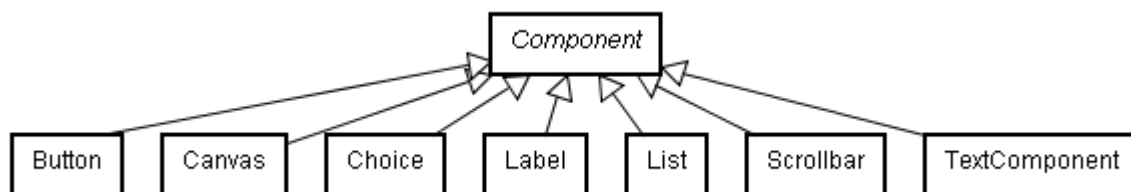
# 1 Interfaces Gráficas com Java

## 1.1 AWT (Abstract Window Toolkit)

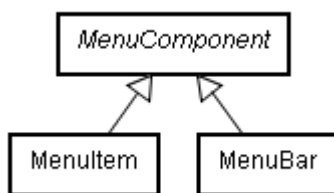
AWT é uma das APIs utilizadas para criação de interfaces gráficas e applets.

Todos os componentes de uma interface gráfica (botões, caixa de combinação etc.) são subclasses de `Component` ou `MenuComponent`.

Um `Component` é um objeto que pode ser representado graficamente em uma tela, além de interagir com o usuário. A classe abstrata `Component` é a superclasse de todas as classes não relacionadas a menu em AWT.



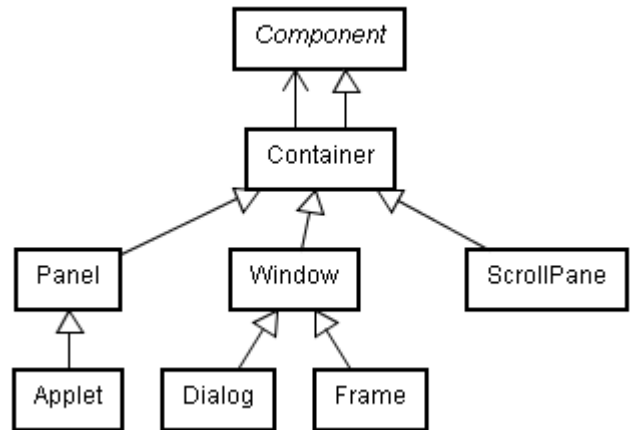
A classe abstrata `MenuComponent` é a super classe de todos os componentes relacionados a menu.



### 1.1.1 Janelas

Na hierarquia de classes, há uma classe chamada `Container`, que representa um objeto que poderá conter outros, como, por exemplo, `Panel` e `Window` (janelas gráficas) que, por sua vez, podem conter outros componentes.

- **Panel:** É o container mais simples de todos, apenas permitindo apenas que outros componentes sejam adicionados. Representa uma "sub-área" de nossa janela, onde a aplicação pode colocar qualquer outro componente, incluindo outros objetos do tipo `Container`.
- **Applet:** Este container deve ser visualizado através de um browser, ou então, do utilitário `appletviewer`, que é distribuído juntamente com o J2SE.
- **Window:** janelas sem borda ou menu, que depende de outra janela para ser exibida.
- **Dialog:** utilizado para apresentações de diálogos.
- **Frame:** janelas de aplicações convencionais que contêm bordas para redimensionamento, botões de maximizar e minimizar, etc.
- **ScrollPane:** janelas de aplicação que possuem barras para navegação (scroll) horizontais ou verticais, podendo ser configuradas para aparecer sempre, nunca ou conforme necessário.



Anotações

### 1.1.2 Criando uma janela simples

Podemos criar uma janela estendendo a classe `Frame`, configurando seu tamanho e chamando o método `setVisible(true)` para que a janela seja exibida.

#### Exemplo: `Frame1.java`

---

```
package br.com.globalcode.awt;
```

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

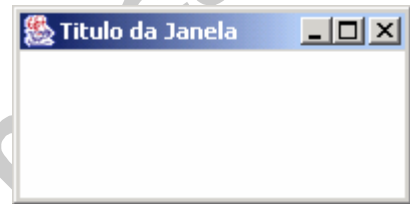
```
public class Frame1 extends Frame {
```

```
    public Frame1(String title) {  
        // Estamos indicando o titulo da janela  
        setTitle(title);  
        // Configuracao do tamanho da janela  
        setSize(200, 100);  
        // Fazendo a janela aparecer  
        setVisible(true);  
    }
```

```
    // Ao executar a classe, o metodo main sera executado, criando uma instancia  
    // do Frame1 que, por sua vez, fará com que a janela seja exibida.
```

```
    public static void main(String args[]) {  
        Frame1 janela = new Frame1("Titulo da Janela");  
    }
```

```
}
```



#### Importante:

Reparem que estas janelas não tem comportamento padrão quando o usuário clica no botão X localizado no canto superior direito. A janela simplesmente não fecha porque é necessário programar um evento que será executado quando o usuário clicar no botão.

### 1.1.3 Adicionando um componente ao container

A adição de qualquer componente (ou até mesmo outro Container) AWT é feita utilizando o método `add(Component c)` herdado da classe `Container`.

No exemplo abaixo, iremos utilizar `Label`, um dos componentes mais simples e que representa um texto em uma única linha, que não pode ser editado pelo usuário.

Um `Label` pode ser construído da seguinte forma:

```
Label label1 = new Label("Texto do label");
```

#### Exemplo: FrameComLabel.java

```
package br.com.globalcode.awt;

import java.awt.*;

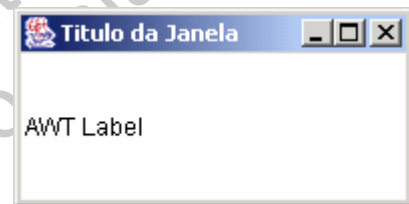
public class FrameComLabel extends Frame {

    public FrameComLabel(String title) {
        // Estamos indicando o titulo da janela
        setTitle(title);
        // Configuracao do tamanho da janela
        setSize(200, 100);

        Label l = new Label("AWT Label");
        add(l);

        // Fazendo a janela aparecer
        setVisible(true);
    }

    // Ao executar a classe, o metodo main sera executado, criando uma instancia
    // do Frame1, que por sua vez faz com que a janela seja exibida.
    public static void main(String args[]) {
        FrameComLabel janela = new FrameComLabel("Titulo da Janela");
    }
}
```



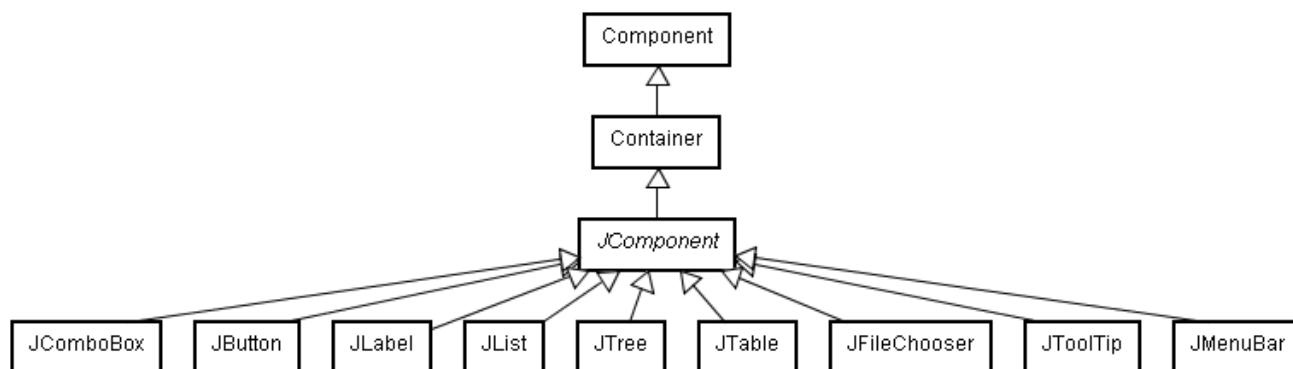
#### Anotações

## 1.2 Swing

Swing é a API mais recente para desenvolvimento de interfaces gráficas baseada em AWT, mas muito mais rica em componentes.

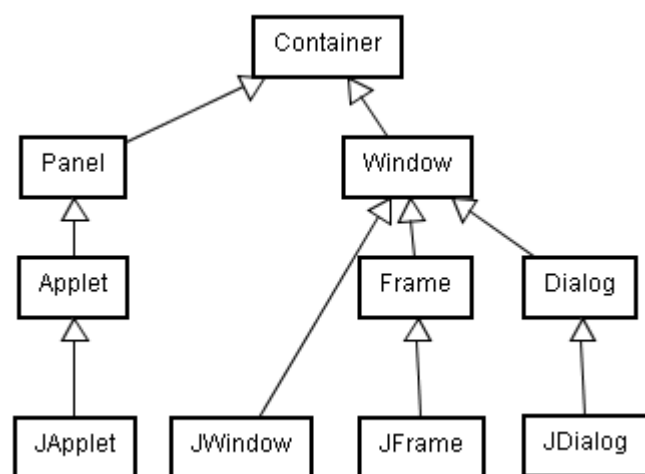
Esta API não depende de código nativo para representar botões, janelas e outros componentes gráficos, sendo totalmente independente de plataforma e desenhando seus próprios componentes.

A classe `JComponent` é a superclasse de todos os componentes que não são containers de outros, sendo derivada de `Component`, como podemos observar no seguinte diagrama de classes:



### 1.2.1 Containers

Vejamos o diagrama abaixo:





## 1.2.2 Criando uma janela simples

### Exemplo: FrameSwing1.java

```
package br.com.globalcode.swing;
```

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

```
public class FrameSwing1 extends JFrame {
```

```
    public FrameSwing1() {
        super("Janela Swing");
        setSize(275, 100);
        show();
    }
```

```
    public static void main(String args[]) {
        FrameSwing1 t = new FrameSwing1();
    }
```

#### Importante:

Esta janela possui comportamento padrão `HIDE_ON_CLOSE` no botão fechar (X no canto superior direito), ou seja, quando clicamos, a janela é "escondida", mas o processo continua sendo executado. Recomendamos configurar o comportamento de fechamento default, utilizando o seguinte método da classe `JFrame`, que provoca o término da execução.:

```
setDefaultCloseOperation(EXIT_ON_CLOSE);
```



### Exemplo: FrameSwing2.java

```
package br.com.globalcode.swing;
```

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

```
public class FrameSwing2 extends JFrame {
```

```
    public FrameSwing2() {
        super("Janela Swing");
        // Utilizamos a constante declarada na classe JFrame para definir
        // o comportamento padrao no fechamento da janela
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(275, 100);
        show();
    }
```

```
    public static void main(String args[]) {
        FrameSwing2 t = new FrameSwing2();
    }
}
```

#### Anotações

### 1.2.3 Adicionando um componente ao container

A manipulação e adição de componentes no `JFrame` é um pouco diferente da que vimos em AWT, pois nunca manipulamos diretamente o `JFrame`. Devemos sempre obter uma referência para o único `Container` contido no `JFrame`, obtido através da chamada ao método `getContentPane()`.

No exemplo abaixo, iremos utilizar o `JLabel`, um dos componentes mais simples e que representa o texto em uma única linha, não podendo ser editado pelo usuário.

Um `JLabel` pode ser construído da seguinte forma:

```
JLabel label1 = new JLabel("Texto do jlabel");
```

#### Exemplo: `FrameSwingComLabel.java`

```
package br.com.globalcode.swing;
```

```
import java.awt.Container;
```

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

```
public class FrameSwingComLabel extends JFrame {
```

```
    public FrameSwingComLabel() {
```

```
        super("Janela Swing");
```

```
        // Utilizamos a constante declarada na classe
```

```
        // JFrame para definir comportamento padrao
```

```
        // no fechamento da janela
```

```
        setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
        JLabel label = new JLabel("Texto do JLabel");
```

```
        Container c = this.getContentPane();
```

```
        c.add(label);
```

```
        setSize(275, 100);
```

```
        show();
```

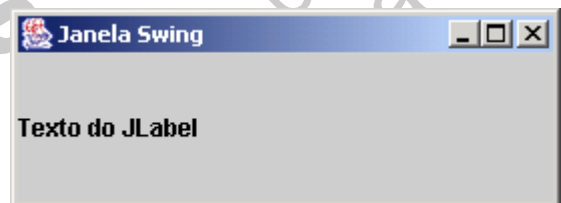
```
    }
```

```
    public static void main(String args[]) {
```

```
        FrameSwingComLabel t = new FrameSwingComLabel();
```

```
    }
```

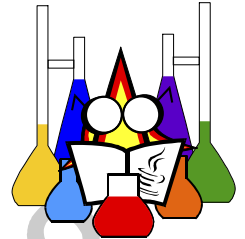
```
}
```



## 1.2.4 Laboratório 1

### Objetivo:

Praticar a criação de janelas.



LABORATÓRIO

### Tabela de atividades

Atividade	OK
1. Faça o download do arquivo <b>aj4lab01_01.zip</b> na URL indicada pelo instrutor(a).	
2. Descompacte o arquivo em seu diretório de trabalho.	
3. Compile e teste as classes <code>Frame1.java</code> e <code>FrameSwing.java</code> .	

Anotações

---



---



---



---



---



---



---

## 1.3 Principais Componentes Swing

### 1.3.1 javax.swing.JButton

Representa um botão com rótulo.

#### Construtores

- ➔ **JButton(String texto):** o valor da variável texto será apresentado no botão;
- ➔ **JButton(Icon icone):** o ícone será apresentado no botão.

**Observação:** Uma das formas de criarmos um ícone é a partir da classe ImageIcon, que pode ser construída a partir de uma String representando o nome da imagem.

Exemplo: ImageIcon icone = new ImageIcon("icone.gif")

Para mais detalhes consulte o Javadoc.

#### Métodos

- ➔ **void setMnemonic(int mnemonic) :** Podemos associar atalhos aos botões utilizando este método, que recebe um int como parâmetro, representado pelas constantes definidas na classe KeyEvent.

#### Exemplos de constantes:

KeyEvent.VK\_A => Atalho para A

KeyEvent.VK\_B => Atalho para B

KeyEvent.VK\_1 => Atalho para 1

#### Exemplo: TesteJButton.java

```
package br.com.globalcode.swing;
import java.awt.*;
import javax.swing.*;
import java.awt.event.KeyEvent;

public class TesteJButton extends JFrame {

    public TesteJButton() {
        super("Teste JButton");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container c = getContentPane();

        JButton botaoOk = new JButton("Ok");
        botaoOk.setMnemonic(KeyEvent.VK_O);
        c.add(botaoOk);

        setSize(200, 50);
        show();
    }

    public static void main(String args[]) {
        TesteJButton t = new TesteJButton();
    }
}
```



**Exemplo: TesteJButtonComIcone.java**

```
package br.com.globalcode.swing;
```

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

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

```
public class TesteJButtonComIcone extends JFrame {
```

```
    public TesteJButtonComIcone() {
```

```
        super("Teste JButton");
```

```
        setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
        Container c = getContentPane();
```

```
        // Criacao de um icone com uma imagem
```

```
        ImageIcon iconeBotao = new ImageIcon("duke.gif");
```

```
        // Criacao de um botao com o icone iconeBotao
```

```
        JButton botaoIcone = new JButton(iconeBotao);
```

```
        // Alteramos a cor de fundo do botao para ficar compativel com a imagem
```

```
        botaoIcone.setBackground(Color.WHITE);
```

```
        c.add(botaoIcone);
```

```
        setSize(80, 150);
```

```
        show();
```

```
    }
```

```
    public static void main(String args[]) {
```

```
        TesteJButtonComIcone t = new TesteJButtonComIcone();
```

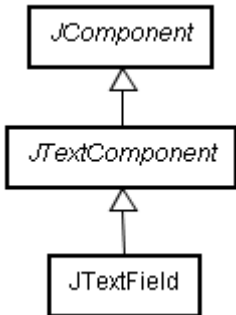
```
    }
```

```
}
```

**Anotações**

### 1.3.2 javax.swing.JTextField

Esta classe representa um campo de texto digitável, usualmente empregado para campos de cadastro de uma única linha.



#### Exemplo: TesteJTextField.java

```

package br.com.globalcode.swing;

import java.awt.*;
import javax.swing.*;

public class TesteJTextField extends JFrame {

    public TesteJTextField() {
        super("Teste JTextField");
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container c = getContentPane();

        JTextField textField = new JTextField();
        c.add(textField);

        setSize(200, 50);
        show();
    }

    public static void main(String args[]) {
        TesteJTextField t = new TesteJTextField();
    }
}
  
```



#### Métodos para manipulação do conteúdo de um JTextField

- ➔ **String getText():** retorna o valor do texto contido no JTextField;
- ➔ **void setText(String texto):** atribui o valor da variável texto ao JTextField;
- ➔ **void setEditable(boolean editable):** habilita ou desabilita o componente de texto. Este método é herdado da classe JTextComponent.

### 1.3.3 javax.swing.JComboBox

#### Construtores

---

- ➔ `JComboBox();`
- ➔ `JComboBox(Object[] itens);`
- ➔ `JComboBox(Vector itens);`

#### Principais métodos

---

- ➔ `void addItem(Object o):` adiciona um item na ultima posição;
- ➔ `Object getSelectedItem():` retorna o item selecionado;
- ➔ `void insertItemAt(Object item, int posicao):` insere um objeto na posição especificada;
- ➔ `Object getItemAt(int posição):` retorna o item que estiver na posição especificada ou null se ele não existir;
- ➔ `void removeAllItems():` remove todos os itens do JComboBox;
- ➔ `void removeItemAt(int posicao):` remove o item que estiver na posição especificada;
- ➔ `void setEnabled(boolean habilitado):` habilita ou não o JComboBox;
- ➔ `void setSelectedItem(Object item):` configura qual será o item selecionado;
- ➔ `void setSelectedIndex(int posicao):` configura qual será o item selecionado através da sua posição no JComboBox.

O componente JComboBox apresenta uma lista com scroll de itens, podendo ser configurado para que cada elemento possa ser editável.

É possível construirmos um JComboBox passando como parâmetro um Vector, contendo os elementos que queremos exibir no JComboBox; o texto exibido será o resultado da chamada ao método `toString` de cada componente.

Neste exemplo, estamos adicionando objetos do tipo `String` ao JComboBox através do método `addItem(String)`.

#### Anotações

---



---



---



---



---



---



---