UNIVERSIDADE FEDERAL DE UBERLÂNDIA

# Estudo Dirigido e Desenvolvimento de Aplicações OO com Interfaces Gráficas em Java e UML

Gabriel Nakamura
João Guilherme
Luís Gustavo
Pedro Paulo

Monte Carmelo
2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

# Relatório da Atividade Avaliativa-1 da disciplina de Programação Orientada a Objetos I

Relatório sobre a AAE-1 que teve como objetivo orientar os alunos na construção de interfaces gráficas usando Java e seus pacotes gráficos tradicionais AWT e Swing, bem como introduzir e praticar a representação de modelos OO em UML.

Gabriel Nakamura

João Guilherme

Luís Gustavo

Pedro Paulo

# Sumário

# 1. Introdução

Aos alunos foi proposta a seleção criteriosa de componentes pertencentes aos pacotes gráficos Swing e AWT da linguagem Java, visando aprofundar o conhecimento sobre as capacidades dessas ferramentas. O objetivo era explorar detalhadamente cada componente, analisando sua aplicação em projetos específicos e desenvolvendo os diagramas UML correspondentes.

Dessa forma, os estudantes foram incentivados a mergulhar na prática do desenvolvimento de interfaces gráficas, compreendendo não apenas a implementação técnica, mas também a lógica por trás de cada elemento escolhido. Além disso, a elaboração dos diagramas UML proporcionou uma visão abrangente do design orientado a objetos, integrando teoria e prática de forma significativa.

## 1.1 Apresentação do Tema

Os itens selecionados pelo grupo foram:

- **Componentes:**

  - Buttons
  - Labels
  - Combo Boxes

- **Contêiner e Painéis:**

  - Panels

- **Gerenciadores de Layout:**

  - Grid Layout

# 1.2 Atividade Proposta

Em suma, foi proposto pelo Professor a resolução de cada um dos tópicos a seguir, para cada item atribuído ao grupo:

1) Criação de diagramas de classes UML detalhando:

- Classes relacionadas ao item selecionado
- Atributos, métodos e construtores
- Anotações descrevendo responsabilidades, importância de atributos e funções dos métodos

2) Elaboração de diagramas de classes UML mostrando relações com outras classes, incluindo superclasses, subclasses, interfaces e outros tipos de relações.

3) Exemplos práticos de uso do item, destacando sua utilidade em programas e suas principais funcionalidades.

4) Demonstração prática do item através da criação de projetos, apresentando código, implementação e relação com os diagramas UML anteriores.

5) Criação de diagrama de sequência ilustrando cenário de uso do item e sua interação com objetos de classes diferentes.

6) Desenvolvimento de diagrama de pacotes para visualizar relações entre os pacotes envolvidos nos tutoriais, destacando dependências entre as classes utilizadas.

# 1.3 Orientações

Por questões de organização, no relatório, foi proposta a inserção dos códigos referentes a cada item, bem como os diagramas UML de classes e interfaces, em links. Ao clicar nos links, o leitor será direcionado para o respectivo conteúdo. Contudo, alguns dos diagramas encontram-se ao final do relaório.

# 2. Buttons

**1.**

**a.**

As classes diretamente ligadas ao item JButton são java.swing.AbstractButton, BasicArrowButton e MetalComboBoxButton.

**b.**

Métodos importantes da classe `javax.swing.AbstractButton` incluem `addActionListener(ActionListener listener)` para adicionar ouvintes de ação ao botão e `setIcon(Icon icon)` para definir o ícone do botão e os atributos text, e icon.

Na classe `BasicArrowButton`, destacam-se o método `paintComponent(Graphics g)`, que é responsável por desenhar o botão, e o método `setDirection(int direction)`, que define a direção da seta do botão.

Já a classe MetalComboBoxButton tem como principais métodos setIconOnly(boolean isIconOnly), e getMinimumSize().

**c.**

**I)** Seus modificadores de visibilidade aos atributos são privados e protegidos, já aos métodos geralmente públicos. Todos os métodos e atributos públicos podem ser acessados diretamente após a criação de uma instância das classes.

**II)** Sim possuem , como por exemplo os getters: getAction(): Retorna o Action associado ao JButton; getIcon(): Retorna o ícone padrão do botão; getText(): Retorna o texto do botão; getToolTipText(): Retorna o texto da dica de ferramenta do botão e etc.

E os setters: setAction(Action a): Define o Action para o JButton; setIcon(Icon defaultIcon): Define o ícone padrão para o botão; setText(String text): Define o texto do botão; setToolTipText(String text): Define o texto da dica de ferramenta que é exibido quando o cursor é posicionado sobre o botão.

**III)** JButton em Java possui métodos reescritos (ou sobrescritos) e sobrecarregados, muitos dos métodos são reescritos da superclasse AbstractButton, JComponent, Container, e Component. Por exemplo, o método paintComponent(Graphics g) é reescrito para desenhar o botão na tela.

Jbutton(), pode ser sobrecarregado para permitir criar um botão de várias maneiras diferentes.

**IV)** Não possui métodos estáticos, a maioria dos métodos e atributos usados pertencem a uma instância específica.

**d.**

**I)** Buttons possui vários construtores sobrecarregados, que permitem criar um botão de várias maneiras diferentes, são eles:

**II)** - JButton(): Cria um botão sem texto ou ícone.
- JButton(Action a): Cria um botão onde as propriedades do botão são inicializadas a partir do objeto Action especificado.
- JButton(Icon icon): Cria um botão com um ícone inicial.
- JButton(String text): Cria um botão com o texto inicial especificado.
- JButton(String text, Icon icon): Cria um botão com o texto e o ícone.

**e.**

**I)** JButton é um botão interativo que pode exibir textos ou imagens e acionar eventos quando clicado. JLabel é uma área de exibição para texto ou imagens, não reagindo a eventos de entrada. JComboBox permite escolher entre opções em uma lista suspensa ou em um campo de texto. JPanel é um contêiner leve que pode adicionar cores, bordas e personalizações visuais. GridLayout organiza componentes em uma grade retangular de células de tamanho igual.

A principal diferença reside no propósito de cada componente: JButton é interativo, JLabel é puramente para exibição, JComboBox permite seleção de opções, JPanel controla a aparência visual e GridLayout organiza componentes. Enquanto JButton e JComboBox podem interagir com o usuário, JLabel e JPanel são usados para exibir conteúdo ou personalizar a aparência, respectivamente.

**II)** Os atributos de JButton representam várias propriedades que determinam o comportamento e a aparência do botão, como por exemplo o model que é o modelo de dados que representa o estado atual do botão. Ele contém informações sobre se o botão está habilitado, pressionado, armado (pronto para ser pressionado), selecionado etc. Ou o actionListener , ouvinte que é notificado cada vez que uma ação ocorre. Isso é usado para responder a eventos de clique do botão.

**III)** model: Tipo ButtonModel, interface que define os métodos que todo modelo de botão deve ter.
actionListener: Tipo ActionListener, interface que define os métodos que todo ouvinte de ação deve ter.

changeListener: Tipo ChangeListener, interface que define os métodos que todo ouvinte de mudança deve ter.
ItemListener: Tipo ItemListener, classe que fornece uma lista pra gerenciar todos os ouvintes de evento.
Ui: Tipo ButtonUI, classe que fornece a aparência do botão.

**2.**

**a.**
Sua super classe é a javax.swingAbstractButton

**b.**
Tem como subclasses diretas BasicArrowButton e MetalComboBoxButton.

**c.**
Implementa as seguintes interfaces : ImageObserver, ItemSelectable, MenuContainer, Seriazable, Accessible e Swing Constants.

**d.**
Tem relação direta com uma classe abstrata AbstractButton.

**e.**
Existe diversas relações possíveis que podemos trabalhar Buttons. Um exemplo seria de dependência com ActionListener, pois Jbutton usa a inreface ActionLinester para lidar com eventos de clique.

**3.**

**a.** JButton pode ser útil em um programa para a interação com o usuário, como por exemplo para entrar e sair em uma página, formulário de entrada de dados, navegação em aplicativos e interação em jogos (play,pause e etc).

**b.** Como por exemplo em um aplicativo várias páginas ou telas podem usar o JButtons para navegar entre as páginas, ou em um aplicativo de questionário se pode ter botões de próximo, anterior para navegar entre as perguntas.

**4. Link para o Código:** MyButton

**5. Link para os diagramas de classe:**

Link Diagrama JButton

Link Diagrama Código JButton

**6. Diagrama representado ao final do relatório.**

# 3. Labels

**1.**

**a.**

Classes diretamente ligadas ao item são JComponent, BasicComboBoxRenderer, DefaultListCellRender, DefaultTableCellRenderer e DefaulTreeCellRenderer.

**b.**

Alguns atributos importantes são, text: O texto que o rótulo deve exibir, icon: O ícone que o rótulo deve exibir, disabledIcon: O ícone exibido quando o rótulo está desativado,horizontalAlignment: O alinhamento horizontal do ícone e do texto.verticalAlignment: O alinhamento vertical do ícone e do texto.

JComponent tem métodos como “setEnable", setForeground(Color fg) e setBackground(Color bg), e como atributo “background”e “foreground”.

Ja a claasse BasicComboBoxRender tem como principais métodos setBorder(Border border), setHorizontalAlignment(int alignment), assim como set background também, e com atributos border e background.

As classes DefaultListCellRender, DefaultTableCellRenderer e DefaulTreeCellRenderer compartilham dos mesmos métodos e atributos.

Outros métodos presentes também: add(Component comp): Adiciona um componente ao contêiner, presente em JComponent.

**c.**

**I)** Para atributos, seus modificadores de acesso são privados e ou protegidos. Os métodos ja são em grande maioria públicos.

**II)** Possuem metódos get e set como: getIcon(), getText(), setIcon(), getPreferredSize

**III)** Há reescrita dos métodos apenas, como por exemplo o método “imageUpdate”e ”paramString”. Mas não há sobrecarga dos métodos

**IV)** Não há métodos estáticos(static) na classe JLabel.

**d.**

**I)** Possui 5 construtores dessa forma ocorre sobrecarga.

**II)** JLabel( Icon image), JLabel(Icon image, int horizontalAlignment), JLabel(String text)...demais contrutores invocam os mesmos parametros.

**e.**

**I)** JLabel tem como principal função exibir informação em formato de texto ou ícones. Oque diferente do item Button por exemplo, não requer a interação com o usuário. É eficaz para exbir informações estáticas.

**II)** O atributo o "icon", a princípio é uma referência para o endereço de uma imagem que vai ser passado para ser representado como ícone, ou uma imagem, na janela. Assim como o texto "text" que armazena o conteúdo a ser exibido.

**III)** setText(String text): Define o texto a ser exibido.
setIcon(Icon icon): Define o ícone a ser exibido.
setFont(Font font): Define a fonte para o texto.
setForeground(Color fg): Define a cor do texto (cor do texto).
setBackground(Color bg): Define a cor de fundo do JLabel.
setHorizontalAlignment(int alignment): Define o alinhamento horizontal do texto.

**2.**

**a.**

Sua superclasse é a **javax.swing.JComponent.**

**b.**

Tem como subclasses **BasicComboBoxRenderer, DefaultListCellRender, DefaultTableCellRenderer e DefaulTreeCellRenderer.**

**c.** Suas interfaces são: **ImageObserver, MenuContainer, Serializable, Accessible,SwingConstants**

**d.**

JLabel e nem suas subclasses são classes Abstratas

**e.**

Existe diversas relações possíveis que podemos trabalhar com JLabel. Um exemplo é de agregação parte de um JFrame que pode ter uma agregação de JLabel, pois um Jframe pode conter vários JLabel, mas os JLabel podem existir independente de um JFrame.

**3.**

**a.**

JLabel vai ser útil em programas que se exige colocar um texto ou ícones de forma estática. Programas como por exemplo onde se exige apenas a impressão de um texto como títulos, rótulos descritivos ou imagens explicativas na interface gráfica.

**b.**

Classe que carrega uma variedade de possibilidades editáveis. Além das principais funcionalidades de inserção de texto e ícones, é possível interagir com esses dois elementos, como por exemplo personalizar a cor das letras, o alinhamento nos eixos x e y da janela, dando maior liberdade ao programador em lapidar a interface devido suas necessidades.

**4. Link para o Código:** MyLabel

**5. Link para os diagramas de classe:**

Link Diagrama JLabel

Link Diagrama código JLabel

**6. Diagrama representado ao final do relatório.**

# 4. Combo Box

**1 .**

**a.**

Além da classe javax.swing.JComponent, não há outras classes diretamente ligadas ao JComboBox.

**b.**

Foreground e background são atributos de cor em componentes gráficos Java. É definido usando os métodos setForeground() e setBackground(), e obter suas configurações usando getForeground() e getBackground().

**c.**

**I)** Seus modificadores de visibilidade aos atributos são privados e protegidos, já aos métodos geralmente públicos e protegidos.

**II)** Sim possui inclusive listados no diagrama de classe, como por exemplo, os métodos “s getSelectedItem”, “getItemCount” e “getEditor”.

**III)** O ComboBox permite a reescrita e sobrecarregar. São os métodos reescritos: “addItem” e sobrecarga o método “removeItemAt”.

**IV)** O ComboBox possui métodos estáticos, são eles: “createStandardComboBoxModel”, “updateUI”, “isPopDownVisible” e entre outros.

**d.**

**I)** O ComboBox possui 4 construtores e com isso pode ocorrer sobrecarga.

**II)** JComboBox(), JComboBox(ComboBoxModel<E> aModel), JComboBox(E[] items),
JComboBox(Vector<E> items).

**e.**

**I)** setSelectedItem(Object anObject): Usado para selecionar um item na lista suspensa com base no objeto fornecido. É importante para definir o item selecionado programaticamente.

getSelectedItem(): Retorna o item atualmente selecionado na lista suspensa. Importante para obter o item selecionado pelo usuário ou programaticamente.

addItem(Object item): Adiciona um item à lista suspensa. Importante para

adicionar itens dinamicamente ao JComboBox.

removeItem(Object item): Remove um item da lista suspensa.

removeAllItems(): Remove todos os itens da lista suspensa. Importante para limpar a lista de itens quando necessário.

getItemCount(): Retorna o número de itens na lista suspensa. Importante para saber quantos itens estão disponíveis.

getItemAt(int index): Retorna o item na posição especificada na lista suspensa. Útil para acessar itens individualmente.

getSelectedIndex(): Retorna o índice do item selecionado na lista suspensa. Importante para saber qual item foi selecionado.

setSelectedIndex(int anIndex): Define o item selecionado com base no índice fornecido. Útil para selecionar um item programaticamente.

isEditable(): Retorna se o JComboBox é editável ou não. Importante para verificar se o JComboBox permite a edição direta do usuário.

**II)** dataModel: Este atributo mantém uma referência ao ComboBoxModel que gerencia os dados exibidos no JComboBox. É fundamental para conectar o JComboBox aos dados que serão exibidos na lista suspensa.

renderer: Armazena uma referência ao ListCellRenderer que determina como os itens na lista suspensa são exibidos. Este atributo é importante para personalizar a aparência dos itens.

isEditable: Indica se o JComboBox é editável ou não. Quando editável, permite ao usuário inserir um valor personalizado. Esse atributo é importante para controlar se a caixa de texto está habilitada para edição.

actionCommand: Armazena uma string que representa o comando de ação associado ao JComboBox. Isso pode ser útil para identificar a origem de eventos de ação gerados pelo JComboBox.

actionListener: Mantém uma lista de ouvintes de ação (action listeners) que serão notificados quando ocorrerem eventos de ação no JComboBox. Esse atributo é importante para permitir que o código responda a ações do usuário, como a seleção de um item na lista.

**III)** Explicado no item I

**2.**

**a.**
Sua superclasse é javax.swing.JComponent

**b.**
Não possui subclasses

**c.**
Suas interfaces são: ActionListener, ImageObserver, ItemSelectable, MenuContainer, Serializable, EventListener, Accessible, ListDataListener

**d.**
O ComboBox é uma classe concreta e por isso que não contém classes abstratas.

**e.**
Existe diversas relações possíveis que podemos trabalhar com o JComboBox, por exemplo, pode se agregar dentro de um FlowLayout, ser composto por um Jpanel e pode tambem depender de um JFrame.

**3.**

**a.**
pode ser útil em um programa que tem diversas opções para mostrar ao usuário e ele organiza em um formato de lista por exemplo uma seleção de opções adicionais para adicionar em um prato de comida

**b.**
Funcionalidade vai de uma seleção de item, os usuários podem escolher um item da lista de opções disponíveis, é um modelo de dados flexível, os itens do ComboBox podem ser preenchidos a partir de um array, uma coleção ou um modelo de dados personalizado

**4. Link para o Código:** Combo Box

**5. Link para os diagramas de classe:**

**Link Diagrama ComboBox**

**Link Diagrama Código ComboBox ,GridLayout e Buttons**

**6. Diagrama representado ao final do relatório.**

# 5. Panels

**1.**

**a.** São elas javax.swing.JComponent e suas subclasses AbstractColorChooserPanel e JSpinner.DefaultEditor

**b.**

Como citado anteriormente para Jcomponent foreground e background são seus atributos de cor. É definido usando os métodos setForeground() e setBackground(), e obter suas configurações usando getForeground() e getBackground().

São métodos que fazem parte da classe AbstractColorChooserPanel:
buildChooser(): Constrói um novo painel de escolha,
getColorFromModel(): Retorna a cor atualmente selecionada,
getColorSelectionModel(): Retorna o modelo que o painel de escolha está editando,
getDisplayName(): Retorna o nome de exibição do painel.

Já da classe JFormattedTextField temos, stateChanged(ChangeEvent e): Chamado quando o valor do modelo do JSpinner muda e propertyChange(PropertyChangeEvent e): Chamado quando uma propriedade do JFormattedTextField muda.

**c.**

**I)** Seus modificadores de visibilidade aos atributos são privados e protegidos, enquanto aos métodos geralmente públicos.

**II)** Sim, possui métodos como “setUI”, “getUIClassID”, “getAccessibleContext” e “getUI”.

**III)** Em JPanel, a reescrita de métodos é mais comum, porém a sobrecarga não é tão frequente. Alguns métodos reescritos incluem: “paintComponent”, “add”, “addLayoutComponent”, “remove” e outros.

**IV)** O JPanel não possui métodos estáticos.

**d.**

**I)** O JPanel possui 4 construtores, portanto há sobrecarga.

**II)** JPanel(), JPanel(boolean isDoubleBuffered), JPanel(LayoutManager layout) e JPanel(LayoutManager layout, boolean isDoubleBuffered).

**e.**

**I)** getAccessibleContext(): Obtém o AccessibleContext associado a este JPanel.

getUI(): Retorna o objeto de aparência (L&F) que renderiza esse componente.

getUIClassID(): Retorna uma string que especifica o nome da classe L&F que renderiza esse componente.
paramString(): Retorna uma representação em string deste JPanel.

setUI(PanelUI ui): Define a aparência do objeto (L&F) que renderiza esse componente.
updateUI(): Redefine a propriedade da UI com um valor da aparência atual.

**II)** Background (background): Tipo Color. Define a cor de fundo do painel.
Border (border): Tipo Border. Define a borda do painel.
Opaque (opaque): Tipo boolean. Indica se o painel é opaco ou não.

**III)** Respondido no item i

**2.**

**a.**

Sua superclasse é javax.swing.JComponent.

**b.**

Não possui subclasses.

**c.**

Suas interfaces são: ImageObserver, MenuContainer, Serializable, Accessible.

**d.**

O JPanel é uma classe concreta e, portanto, não contém classes abstratas.

**e.**

Existem diversas relações possíveis que podem ser trabalhadas com o JPanel, como agregação com um botão, um JLabel, composição dentro de um JFrame e também pode depender de um JFrame.

**3.**

**a.**

Pode ser usado para um layout específico dentro de um programa com diferentes personalizações e outros componentes.

**b.**

Sua principal funcionalidade é ser um componente flexível e versátil, pois pode ser composto por diversos outros componentes, proporcionando uma aplicação mais versátil e variando os layouts do programa.

Alternativas de uso podem incluir a criação de um painel de configuração dentro de uma aplicação, um painel de mensagem e outras funcionalidades.

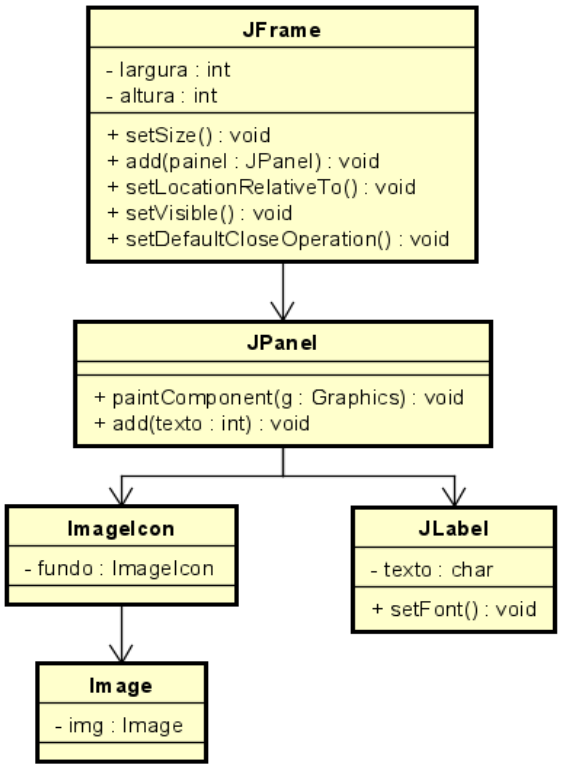
**4. Link para o Código:** MyPanels

**5. Diagramas de classe:**

**i) Diagrama Componente:**

**Link Diagrama Jpanel**

**ii) Diagrama Código:**




**6. Diagrama representado ao final do relatório.**

# 6. GridLayout

**1**.

**a.**

Tem a java.lang.Object a classe diretamente ligada a GridLayout.

**b.**

Lang.Object tem como principais métodos que valem a pena serem destacados:
equals(Object obj): Indica se algum outro objeto é "igual" a este.
hashCode(): Retorna um código hash para o objeto.
toString(): Retorna uma representação de string do objeto.

Como seus atributos getClass() que Retorna a classe do objeto.
wait(): Causa a thread atual a esperar até que outro thread invoque o método notify() ou notifyAll() para esse objeto.

**c.**

**I)** Seus modificadores de visibilidade aos atributos são privados e protegidos, já aos métodos geralmente públicos.

**II)** Sim possui inclusive listados no diagrama de classe, como por exemplo, os métodos “setColumns”, “setRows”, “getColumns” e “getRows”.

**III)** O GridLayout só permite a reescrita em seus métodos, já a sobrecarregados não. São os métodos reescritos: “layoutContainer”, “preferredLayoutSize”, “addLayoutComponent” e entre outros.

**IV)** O GridLayout não possui métodos estáticos.

**d.**

**I)** O GridLayout possui 3 construtores e com isso pode ocorrer sobrecarga..

**II)** GridLayout(), GridLayout(int rows, int cols) e GridLayout(int rows, int cols, int hgap, int vgap)

**e.**

**I)** “addLayoutComponent(String name, Component comp)”: Adiciona o componente especificado com o nome especificado ao layout.

"getColumns()": Obtém o número de colunas neste layout.

"getHgap()": Obtém a lacuna horizontal entre os componentes.

"getRows()": Obtém o número de linhas neste layout.

"getVgap()": Obtém a lacuna vertical entre os componentes.

"layoutContainer(Container parent)": Dispõe o contêiner especificado usando este layout.

"setColumns(int cols)": Define o número de colunas neste layout.

"setRows(int rows)": Define o número de linhas neste layout para o valor especificado.

"toString()" Retorna a representação em string dos valores deste layout de grade.

**II)** Número de Linhas (rows): Tipo int. Define o número de linhas na grade. Número de Colunas (cols): Tipo int. Define o número de colunas na grade. Espaçamento Horizontal (hgap): Tipo int. Define o espaçamento horizontal entre as células da grade. Espaçamento Vertical (vgap): Tipo int. Define o espaçamento vertical entre as células da grade.

**III)** Respondido no item i

**2**.

**a.**
Sua Super Classe é Java Object

**b.**
Não possui subclasses

**c.**
Suas interfaces: LayoutManager, Serializable

**d.**
O GridLayout é uma classe concreta e por isso que não contém classes abstratas.

**e.**
Existe diversas relações possiveis que podemos trabalhar com o GridLayout, por exemplo pode se agregar com um botao, ser composto dentro de um JFrame e pode depender dentro de um JPanel.

**3.**

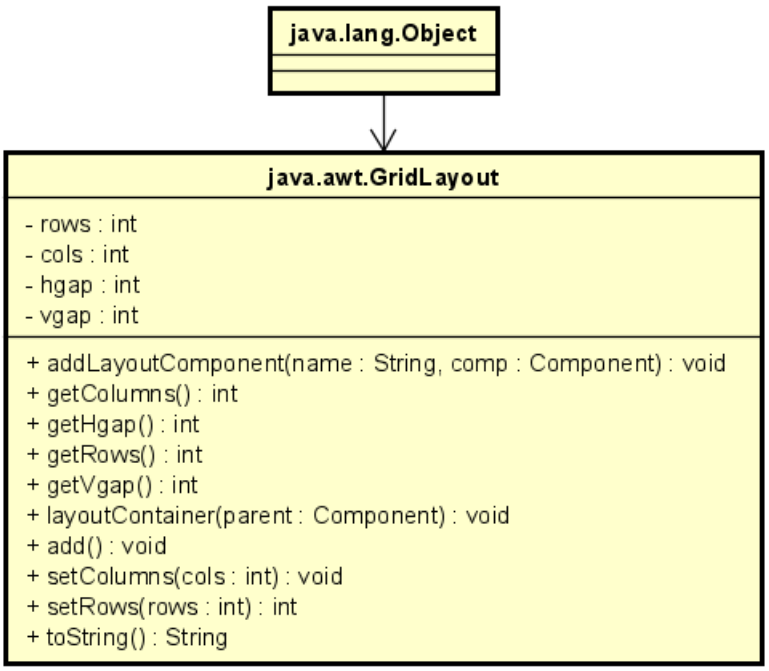
a. Ele pode ser útil quando queremos uma automatização para organizar o layout de vários botões por exemplo, de vez definir coordenada por condenada, utilizamos o GridLayout para realizar essa organização.

b. As funções do GridLayout vai desde organização de componentes em grade, tamanho uniforme, espaçamento uniforme até o dinamismo de uma janela.
Uma alternativa bem usada em relação ao GridLayout é o FlowLayout que o diferencial dele é a organização por linhas, sem depender de colunas.

**4. Link para o Código:** Grid Layout

**5. Diagramas de classe:**

**I) Diagrama Componente:**

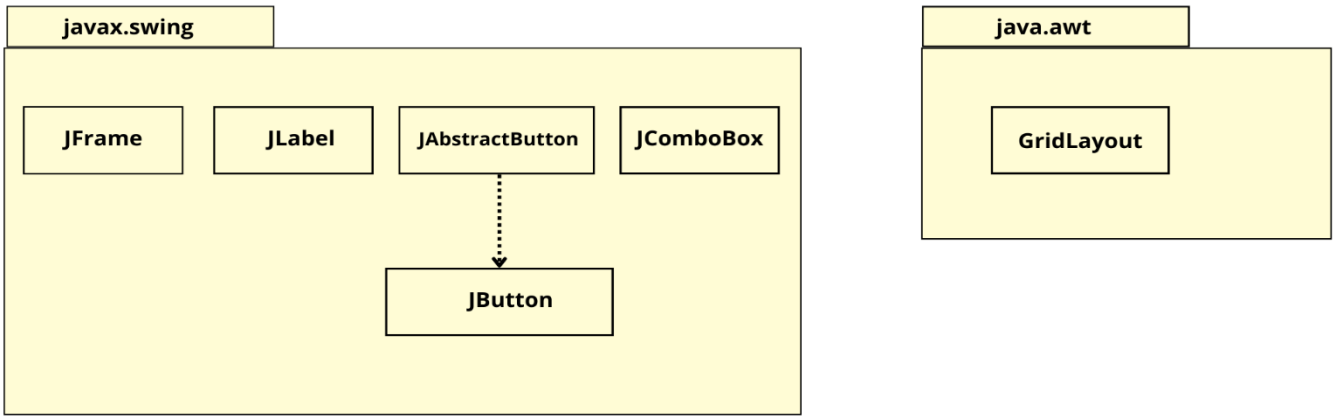



**ii) Diagrama Código: Link Diagrama Código GridLayout, ComboBox, JButton**

**6. Diagrama representado ao final do relatório.**

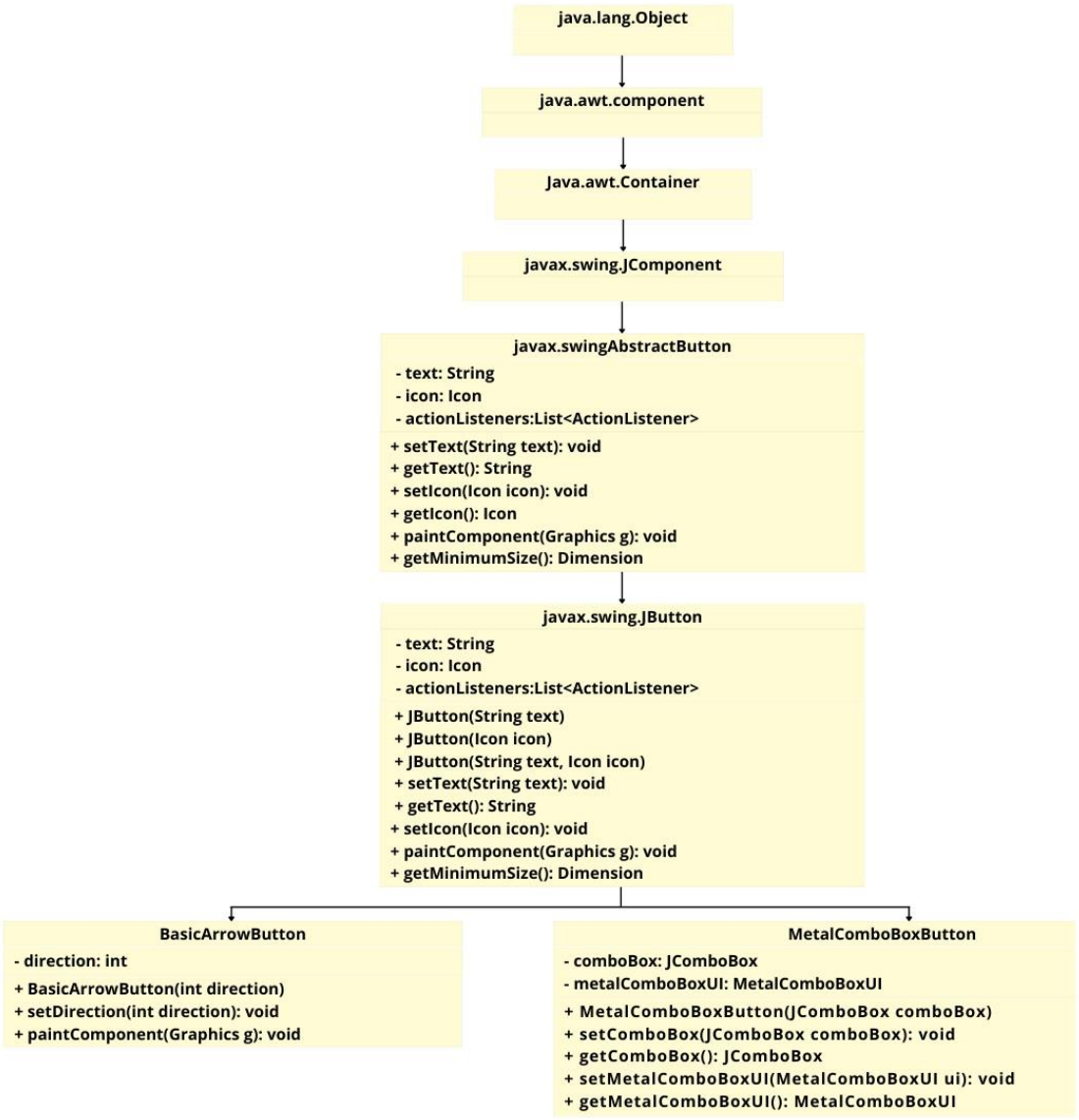
# Diagrama de Pacotes

Diagrama de pacotes dos Itens envolvidos

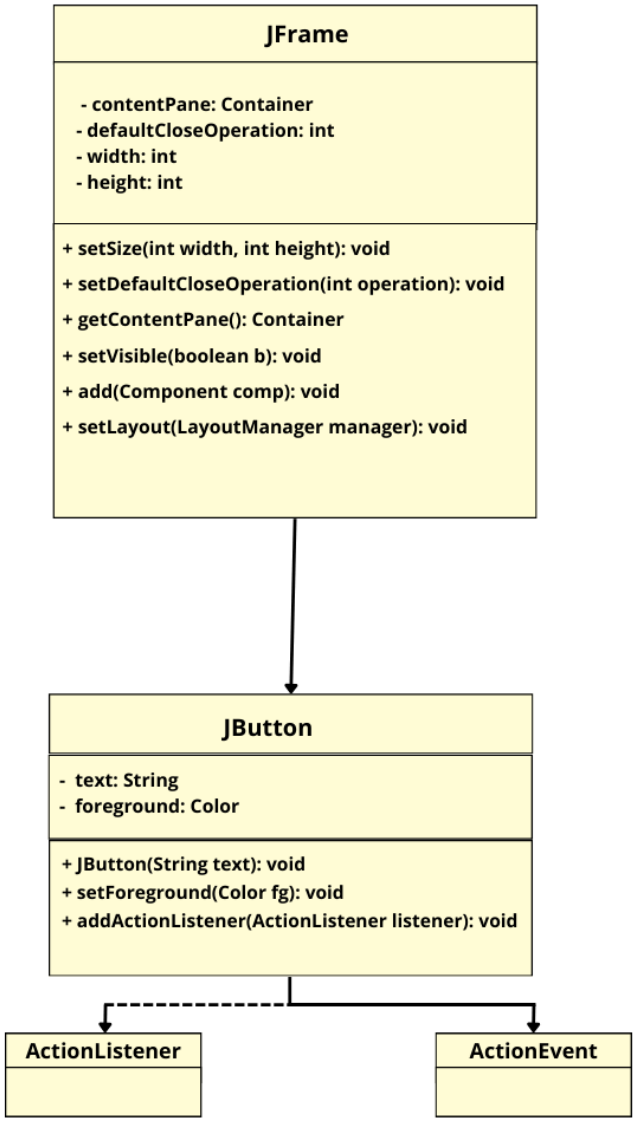


# JButton

## Classe JButton:



# Código JButton :



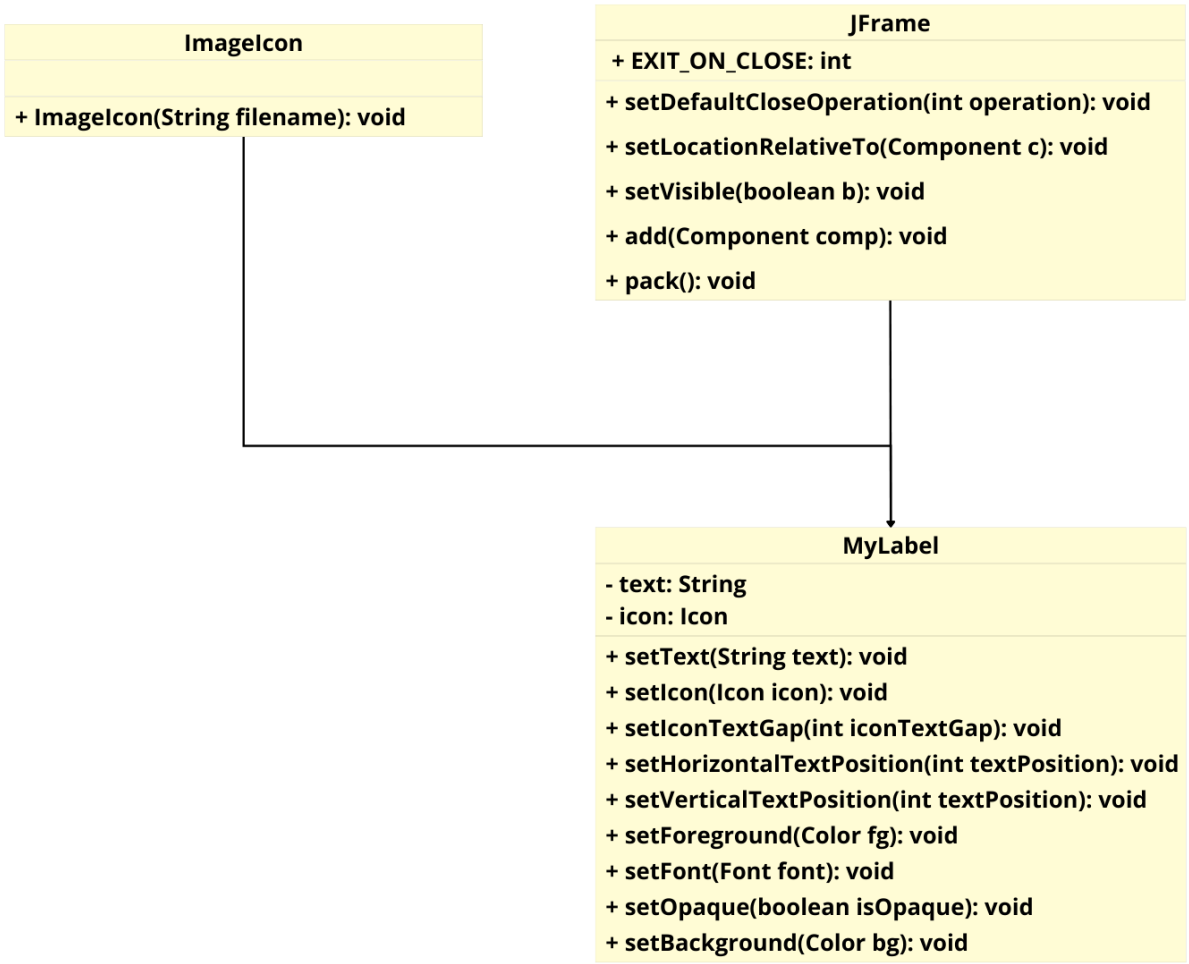
# JLabel

## Classe JLabel :

! Formato fora de tamanho para documento.
Acesse o Link Diagrama JLabel para visualizar.

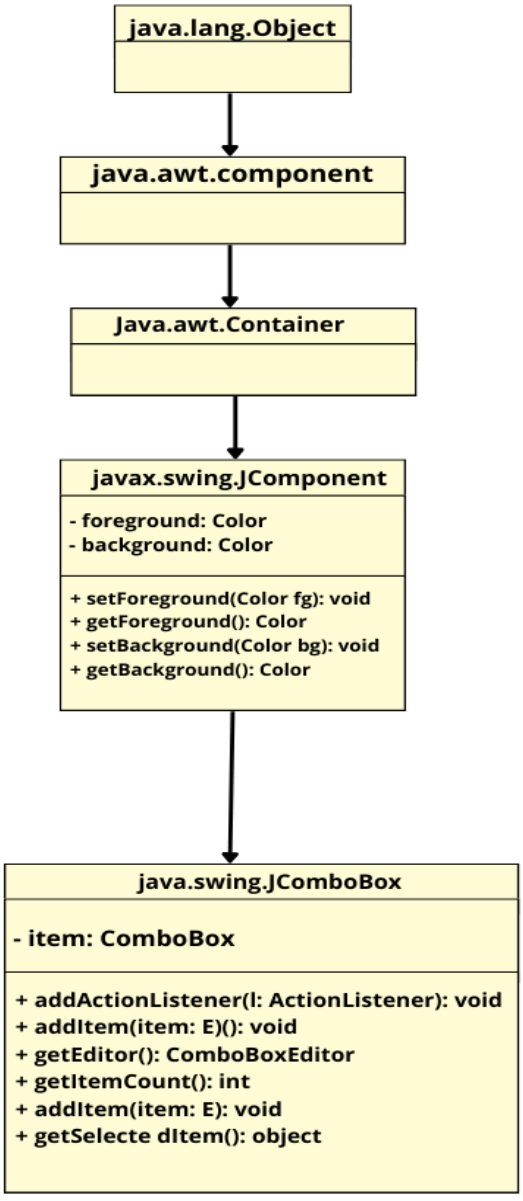
---

## Código JLabel:

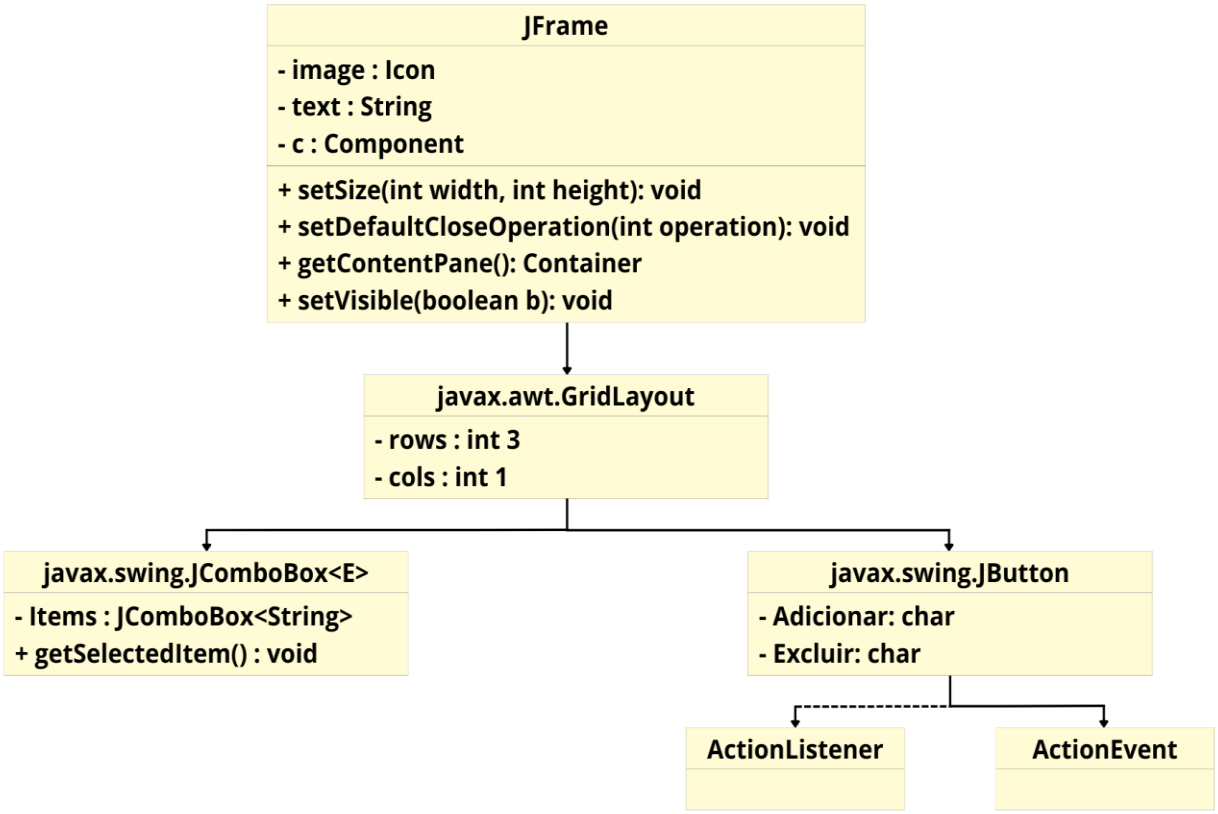


# JComboBox

## Classe JComboBox:

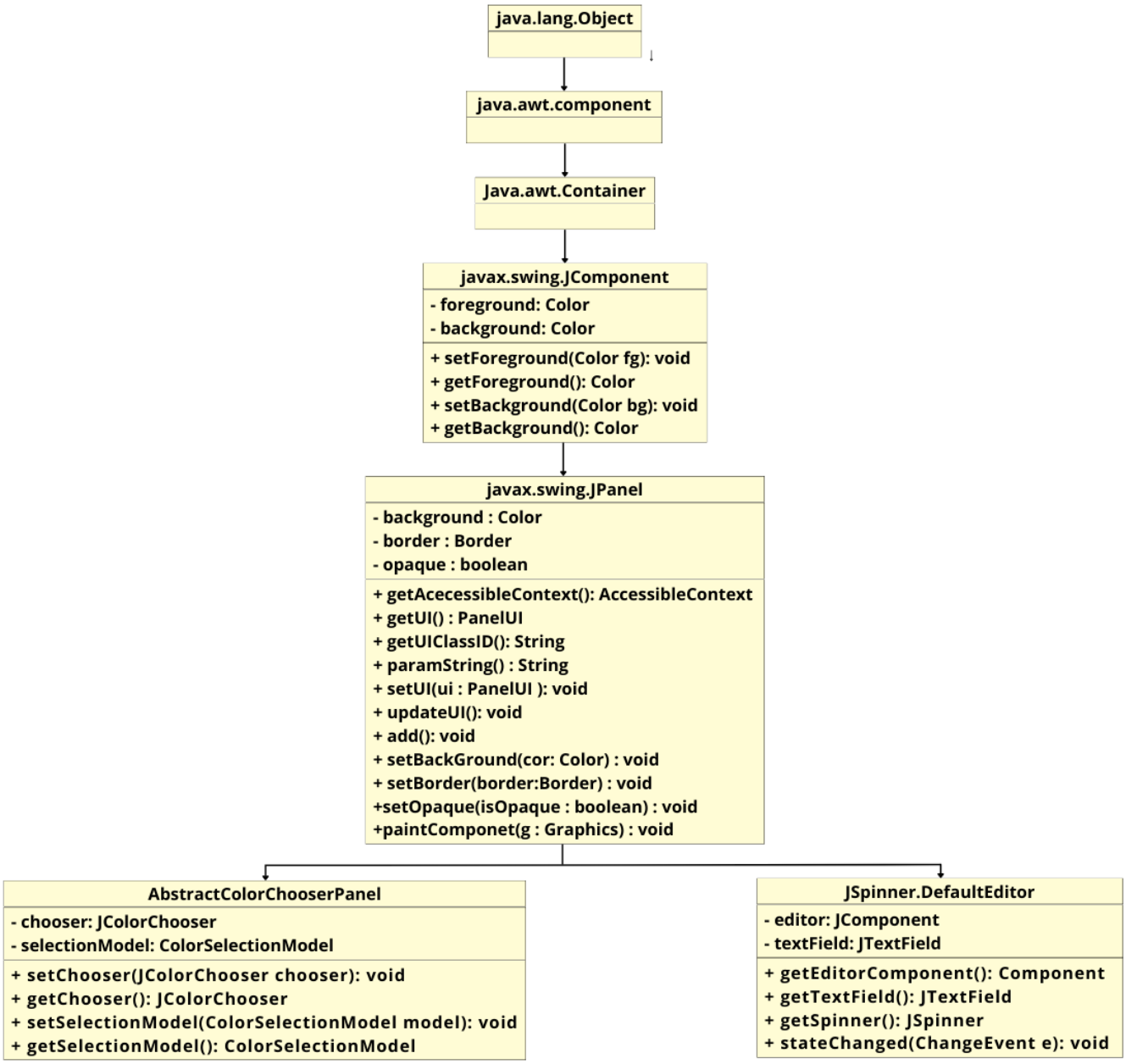


# Código JComboBox:

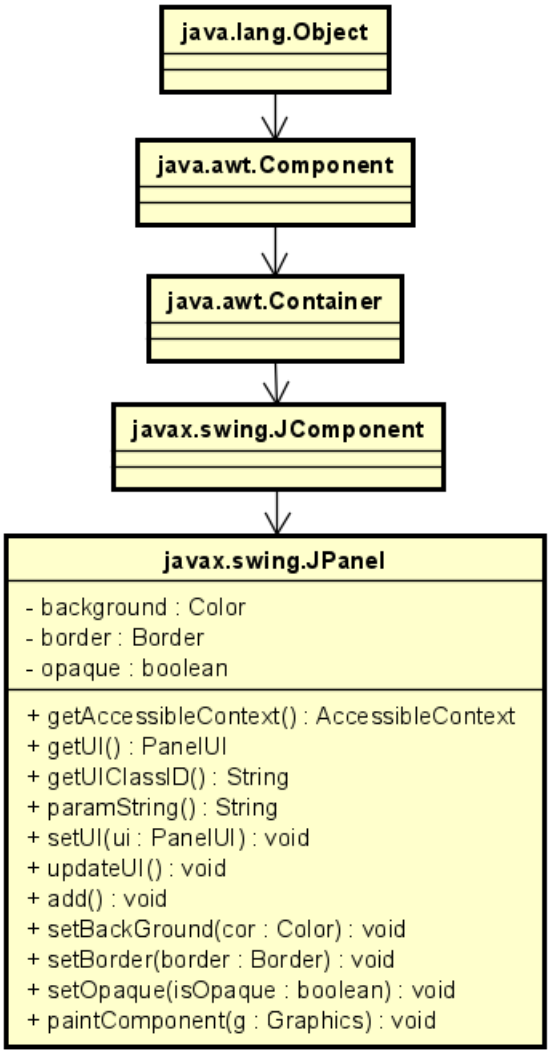


# JPanel

## Classe JPanel :

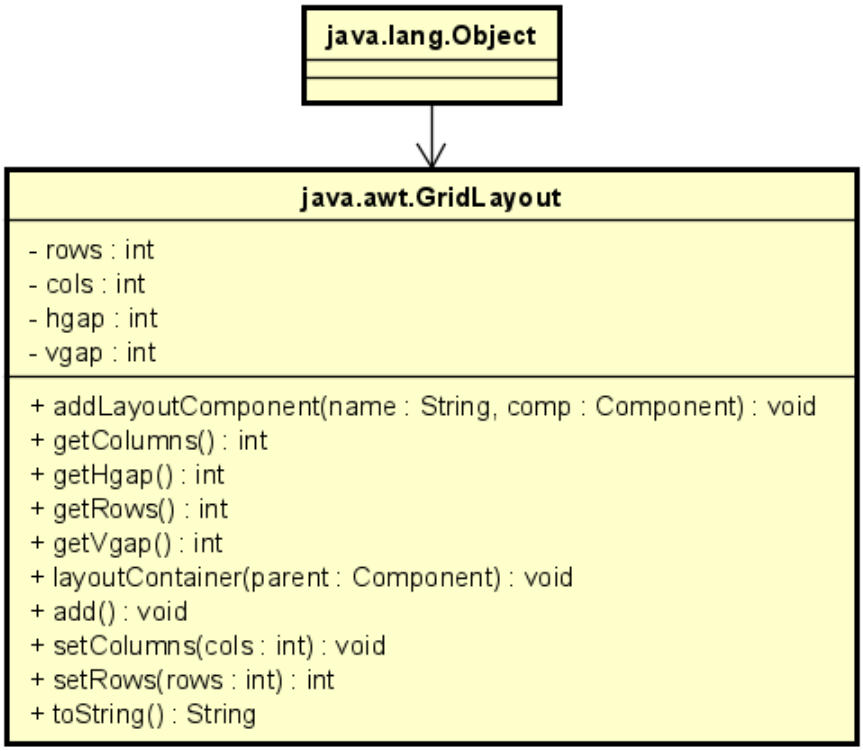


# Código JPanel :

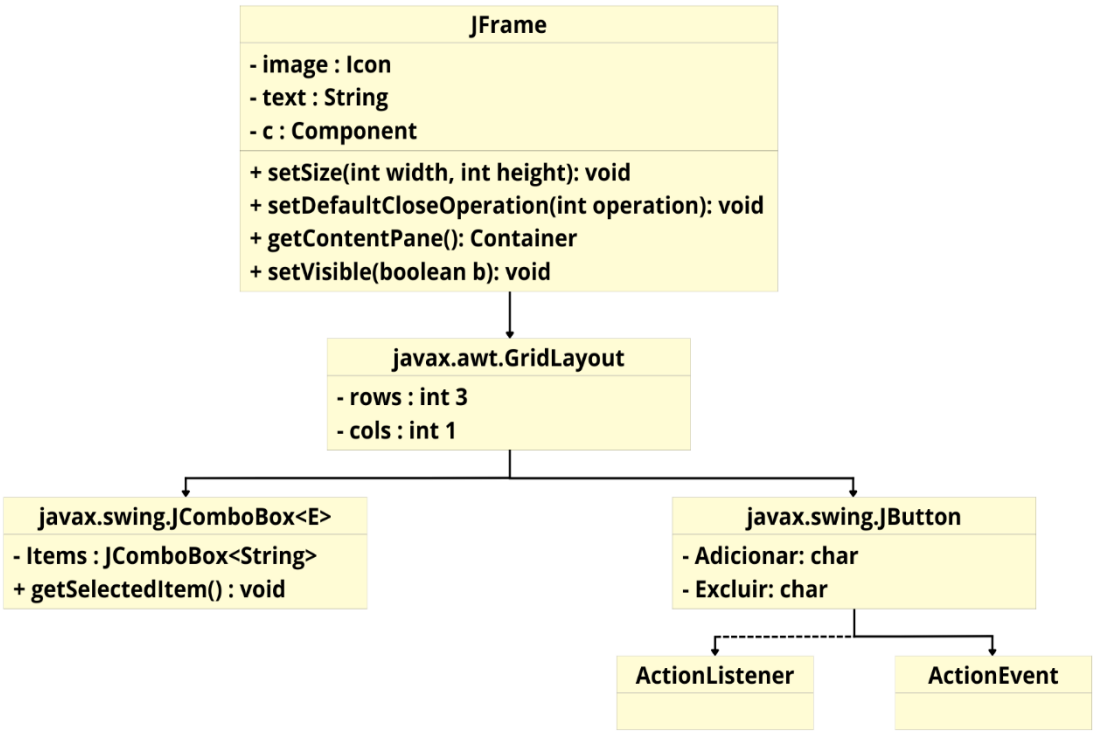


# GridLayout

## Classe GridLayout:



# Código GridLayout :



# Bibliografia


https://docs.oracle.com/javase/tutorial/uiswing/components/label.html

https://docs.oracle.com/javase/8/docs/api/javax/swing/JComponent.html#setForeground-java.awt.Color-

https://docs.oracle.com/javase/8/docs/api/javax/swing/JLabel.html#setIconTextGap-int-

https://www.geeksforgeeks.org/jlabel-java-swing/

https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html

https://docs.oracle.com/javase/8/docs/api/java/awt/Window.html#pack--

https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html#:~:text=An%20abstract%20class%20is%20a,but%20they%20can%20be%20subclassed.&text=When%20an%20abstract%20class%20is,methods%20in%20its%20parent%20class.