Aula 3

Componentes de Interface Gráfica

Java - GUI

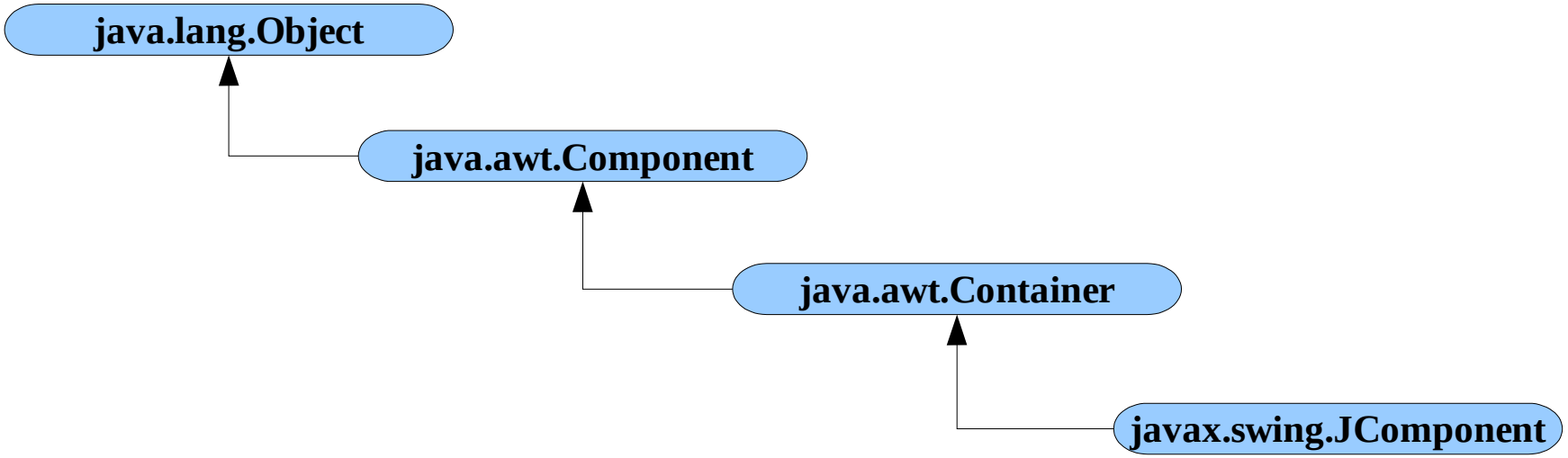
- Definição
  - GUI – Graphical User Interface – Interface Gráfica com o usuário. São construídas a partir de componentes GUI e permitem interação através de eventos (clique mouse, teclado, voz, ...).
- Por que utilizar GUI?
  - Fornece familiaridade com o programa
  - Reduz o tempo de aprendizado
  - Apresenta as informações de forma organizada e compreensível
  - ...

Java - GUI

- Componentes AWT x Componentes Swing
  - Os **componentes AWT** (Abstract Windowing Toolkit) do pacote **java.awt** estão diretamente associados com os recursos da interface gráfica com o usuário da plataforma local. Assim, os componentes são exibidos com uma aparência diferente em cada plataforma;
  - Os **componentes Swing** do pacote **javax.swing** são desenvolvidos totalmente em Java e possibilitam a especificação de uma aparência uniforme para todas as plataformas;
  - Os componentes Swing tornaram-se padrão a partir da plataforma Java 1.2, contudo alguns componentes Swing ainda utilizam o pacote AWT como super classes.

Java - GUI

- Hierarquia dos Componentes Swing



Java - GUI

- JComponent
  - Características: dicas, bordas, aparência interna(cor), look and feel, propriedades personalizadas, suporte para layout, acessibilidade, drag and drop, mapeamentos de teclas.
  - Principais funções
    - personalização de aparência (ex: setFont(), setBorder(), ...)
    - obter e alterar o estado do componente (ex: setEnabled(), setVisible(), ...)
    - tratamento de eventos (ex: addMouseListener, ...)
    - renderização de componentes (ex: repaint(), ...)
    - gerenciamento de hierarquia (ex: getParent(), ...)
    - layout, tamanho e posição (ex: getWidth(), getHeight(), ...)

Java - GUI

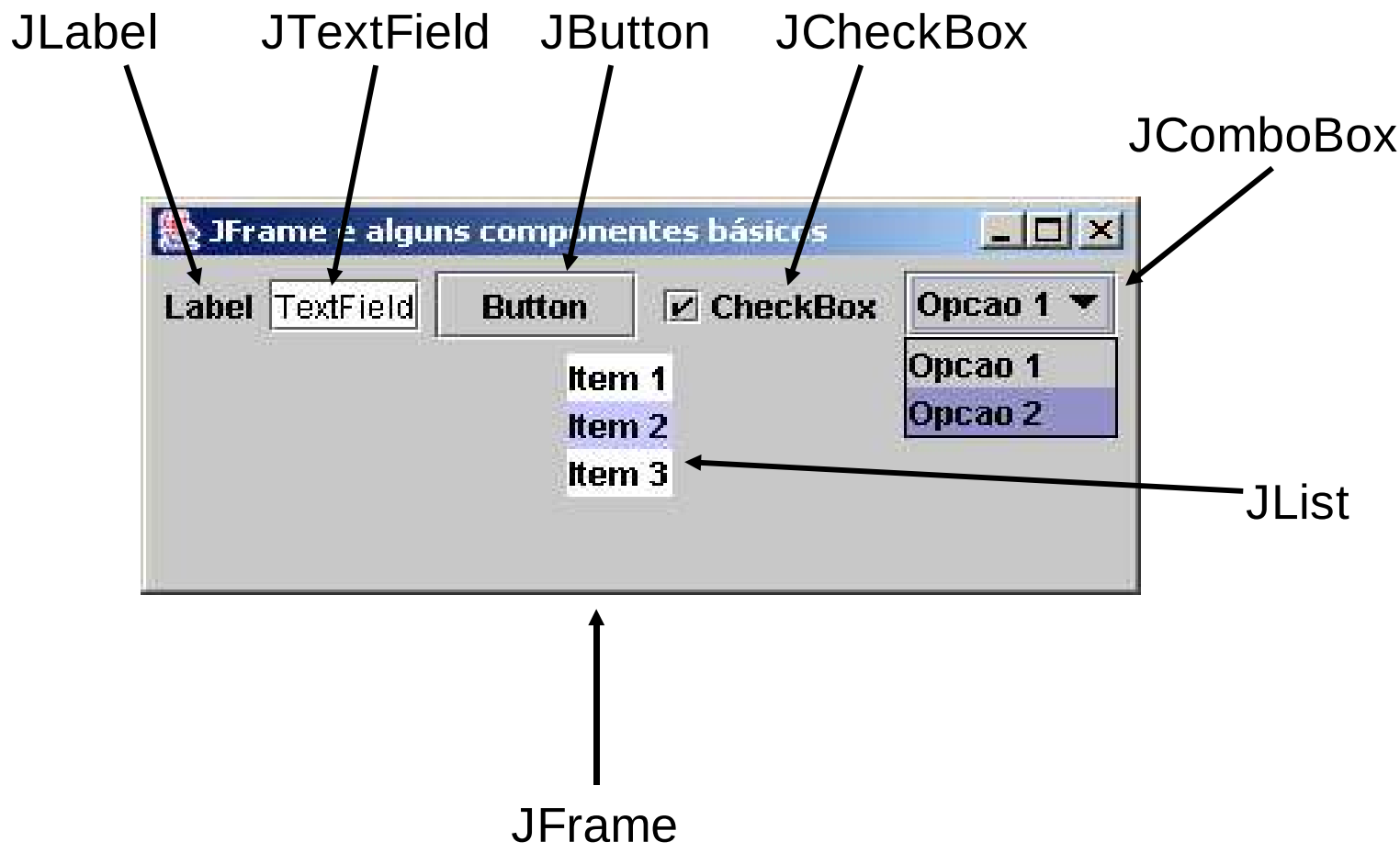
- Hierarquia de *Containers*
  - ***top-level container:*** fornece a área base para que os outros componentes swing possam ser inseridos. (ex: JFrame)
  - ***intermediate container****:* fornece suporte de posicionamento e acessórios para que outros componentes swing possam ser inseridos. (ex: JPanel, JscrollPane, ...)
  - ***atomic components:*** entidades auto-suficientes que representam informações para o usuário. (ex: JTextField)

  - Todo *top-level container* contém indiretamente um *intermediate container* conhecido como *content pane.*

Java - GUI

- Componentes Básicos
  - **JFrame** - É um *container* (formulário) para outros componentes GUI.
  - **JLabel** - Área em que podem ser exibidos texto não-editável ou ícones.
  - **JTextField** - Área em que o usuário insere dados pelo teclado.
  - **JButton** - Área que aciona um evento quando o usuário clica.
  - **JCheckBox** - Possui dois estados: selecionado ou não-selecionado.
  - **JComboBox** - Lista de itens que o usuário pode fazer uma seleção clicando em um item na lista ou digitando na caixa.
  - **JList** - Área em que uma lista é exibida, possibilitando a seleção clicando em qualquer item da lista.
  - **JPanel** - *Container* em que os componentes podem ser colocados.

Java - GUI


JLabel
JTextField
JButton
JCheckBox
JComboBox
JFrame e alguns componentes básicos
Label
TextField
Button
CheckBox
Opcao 1
Opcao 1
Opcao 2
Item 1
Item 2
Item 3
JList
JFrame

Java - GUI

```
...
// Declaração e criação dos componentes
public class FrameSwing extends JFrame {
private JLabel label;
private JTextField textField;
private JButton button;
private JCheckBox checkBox;
private JComboBox comboBox;
private JList list;

private String opcaoCombo[] = {"Opcao 1", "Opcao 2"};
private String itemLista[] =  {"Item 1", "Item 2", "Item 3"};

public FrameSwing() {
super("JFrame e alguns componentes básicos");

label = new JLabel("Label");
textField = new JtextField("TextField");
button = new JButton("Button");
checkBox = new JCheckBox("CheckBox");
comboBox = new JComboBox(opcaoCombo);
list = new Jlist(itemLista);
...
```

Java - GUI

```
/* Container do frame */
Container container = getContentPane();

/* altera o layout */
container.setLayout(new FlowLayout());

/* Adiciona os componentes no container */
container.add(label);
container.add(textField);
container.add(button);
container.add(checkBox);
container.add(comboBox);
container.add(list);

...
```

Java - GUI

- java.awt.Component - Métodos
  - void **add**(PopupMenu popup)
  - boolean **contains**(int x, int y)
  - Color **getBackground**() e void **setBackground**(Color c)
  - Rectangle **getBounds**() e void **setBounds**(int x, int y, int width, int height)
  - Font **getFont**() e void **setFont**(Font f)
  - Color **getForeground**() e void **setForeground**(Color c)
  - Dimension **getPreferredSize**()
  - Dimension **getSize**() e void **setSize**(int width, int height)
  - int **getWidth**() e int **getHeight**()
  - void **setEnabled**(boolean b)
  - void **setLocation**(int x, int y)
  - void **setVisible**(boolean b)

Java - GUI

- java.awt.Container - Métodos
  - Component **add**(Component comp)
  - void **add**(Component comp, Object constraints)
    - Adicionam um componente ao container.
  - Component[] **getComponents**()
    - Retorna um vetor com os componentes agregados no container.
  - LayoutManager **getLayout**() e void **setLayout**(LayoutManager mgr)
    - Informa ou modifica o gerenciador de layout do container
  - void **remove**(Component comp)
    - Remove um componente do container

Java – GUI

- javax.swing.JComponent - Métodos
  - boolean **contains**(int x, int y): verifica se o componente contém esses pontos
  - void **setBackground**(Color bg): altera a cor de fundo
  - void **setBorder**(Border border): altera a borda
  - void **setEnabled**(boolean enabled): habilita ou desabilita o componente
  - void **setFont**(Font font): altera a fonte
  - void **setForeground**(Color fg): altera a primeira camada da cor de fundo
  - void **setOpaque**(boolean isOpaque): deixa o componente opaco
  - void **setPreferredSize**(Dimension preferredSize): configura o tamanho do componente
  - void **setToolTipText**(String text): insere um texto informativo
  - void **setVisible**(boolean aFlag): altera a visibilidade do componente

Java - GUI

- javax.swing.JFrame - Métodos
  - Utilizado geralmente através de herança
    - public class MinhaJanelaPredileta **extends JFrame** {...}
  - **JFrame**() e **JFrame**(String title): Construtores
  - void **setDefaultCloseOperation**(int operation)
    - Altera a operação padrão a ser executada quando o usuário fechar o Jframe
  - public void **setExtendedState**(int state)
    - Altera o estado do JFrame: maximizado, minimizado, ...)
  - void **pack**()
    - Redimensiona a janela para encaixar-se ao tamanho dos seus subcomponentes
  - void **show**(): exibe o janela do Jframe
  - void **setIconImage**(Image image): altera o ícone do canto superior esquerdo
  - void **setResizable**(boolean resizable): define se a janela poderá ser redimensionada
  - void **setTitle**(String title): altera o título da janela

Java - GUI

- javax.swing.JLabel - Métodos
  - **JLabel**(String text) e **JLabel**(String text, int horizontalAlignment)
    - Construtores mais utilizados
  - **JLabel**(Icon image) e **JLabel**(String text, Icon icon, int horizontalAlignment)
    - Construtores com imagem (ícones)
  - String **getText**() e void **setText**(String text)
    - Recupera ou altera o texto
  - void **setHorizontalTextPosition**(int textPosition)
    - Altera o alinhamento horizontal do texto
  - void **setVerticalTextPosition**(int textPosition)
    - Altera o alinhamento vertical do texto

Java - GUI

- javax.swing.text.JTextComponent - Métodos
  - Superclasse direta dos componentes de edição de texto: JTextField, JTextArea e JEditorPane

  - void **copy**(): copia para a área de transferência o texto selecionado
  - void **cut**(): move para a área de transferência o texto selecionado
  - void **paste**(): transfere o conteúdo da área de transferência para o componente
  - String **getSelectedText**(): retorna o texto selecionado
  - int **getSelectionStart**() e int **getSelectionEnd**(): retorna o início e o final da seleção
  - String **getText**() e String **getText**(int offs, int len): retorna o texto
  - void **select**(int selectionStart, int selectionEnd) e void **selectAll**(): seleciona o texto
  - void **setEditable**(boolean b): permite editar ou não o conteúdo do componentevoid
  - void **setText**(String t): altera o texto

Java - GUI

- javax.swing.JButton - Métodos
  - **JButton**(Icon icon); **JButton**(String text) e **JButton**(String text, Icon icon): construtores
  - public void **setMnemonic**(char mnemonic): destaca uma letra para servir como atalho
  - void **setText**(String text): altera o texto do botão
- javax.swing.JCheckBox - Métodos
  - **JCheckBox**(String text);**JCheckBox**(String text, boolean selected); **JCheckBox**(Icon icon)
    - Principais construtores
  - public void **setSelected**(boolean b): habilita ou desabilita a seleção
  - public void **doClick**(): executa o evento referente a um clique no componente
- javax.swing.JRadioButton - Métodos
  - Todos os métodos e construtores são idênticos ao JCheckBox

Java - GUI

- javax.swing.ButtonGroup - Métodos
  - É utilizado com um container para botões, onde somente um botão pode estar ativo em um dado instante
  - **ButtonGroup()**: construtor
  - void **add**(AbstractButton b): adiciona um botão ao grupo

- javax.swing.JComboBox - Métodos
  - **JComboBox**(); **JComboBox**(Object[] items) e **JComboBox**(Vector items): construtores
  - void **addItem**(Object anObject): adiciona um item a lista do combo
  - Object **getItemAt**(int index): retorna o item pelo seu índice
  - int **getItemCount**(): retorna o número de itens
  - Object **getSelectedItem**() : retorna o item selecionado
  - int **getSelectedIndex**(): retorna o índice do item selecionado
  - void **insertItemAt**(Object anObject, int index): insere um item em uma posição específica
  - void **removeItem**(Object anObject): remove o item especificado

Java - GUI

- javax.swing.JList - Métodos
  - **JList**(); **JList**(Object[] listData) e **JList**(ListModel dataModel): principais construtores
  - void **clearSelection**(): limpa a marca de seleção
  - int **getMinSelectionIndex**() e int **getMaxSelectionIndex**(): retorna o mínimo e o máximo índice em uma seleção
  - int **getSelectedIndex**() e int[] **getSelectedIndices**(): retorna o índice dos itens selecionados
  - Object **getSelectedValue**() e Object[] **getSelectedValues**(): retorna os itens selecionados
  - boolean **isSelectionEmpty**(): retorna *true* se não há nada selecionado
  - void **setListData**(Object[] listData): constrói a lista com os valores do parâmetro
- javax.swing.DefaultListModel - Métodos
  - void **add**(int index, Object element) e void **addElement**(Object obj): adiciona um item
  - Object **remove**(int index); void **removeAllElements**() e void **removeElementAt**(int index): remove elementos
  - int **getSize**(): retorna o número de elementos na lista

Java - GUI

- javax.swing.JScrollPane - Métodos
  - Fornece um container com barras de rolagens e cabeçalho vertical e horizontal
  - **JScrollPane**(Component view): principal construtor
  - void **setViewportView**(Component view): altera o componente visualizado
  - void **setViewportBorder**(Border viewportBorder): adiciona uma borda

- javax.swing.JTextArea - Métodos
  - **JTextArea**() e **JTextArea**(int rows, int columns): principais construtores
  - void **append**(String str): adiciona texto após o final do conteúdo
  - void **insert**(String str, int pos): insere o texto em uma posição específica
  - void **setLineWrap**(boolean wrap): altera a quebra de linha
  - void **setText**(String str): substitui todo o conteúdo pelo valor do parâmetro

Java - GUI

- Gerenciamento de *Layout*
  - Definição: é o processo de determinar o tamanho e o posicionamento dos componentes.
  - Todo container tem um gerenciador de layout - um objeto responsável pela organização dos componentes GUI em um container para fins de apresentação.
  - É possível trabalhar sem utilizar um gerenciador de layout, para isso é necessário setá-lo para null.

Java - GUI

- Tipos de Gerenciadores de Layout
  - **FlowLayout:** Organiza os componentes sequencialmente na ordem em que forem adicionados.
  - **BorderLayout:** Organiza os componentes em cinco áreas: norte, sul, leste, oeste e centro.
  - **GridLayout:** Organiza os componentes em linhas e colunas.
  - **BoxLayout:** Organiza os componentes da esquerda para a direita ou de cima para baixo.
  - **CardLayout:** Organiza os componentes em forma de pilha, somente o componente no topo é visível.
  - **GridBagLayout:** Organiza os componentes sobre uma grade, permitindo especificar tamanho, ocupação, posição, ...

Java - GUI

- Processo básico para utilizar layouts:
  - Obter uma referência para o container: **getContentPane()**
  - Criar um novo gerenciador de layout
  - Alterar o layout: **setLayout()**
  - Adicionar os componentes: **add()**

  - Exemplo:

```
....
Container c = this.getContentPane();
BorderLayout bl = new BorderLayout();
c.setLayout (bl);

Jlabel lblTexto = new JLabel( " Teste Layout " );
c.add (lblTexto, BorderLayout.NORTH);
```

Java - GUI

- Modelo de Tratamento de Eventos
  - Eventos são gerados quando o usuário interage com a interface.
  - Os pacotes que implementam eventos são:
    - **java.awt.event** e **javax.swing.event**
  - O mecanismo de tratamento de eventos é composto por:
    - ***event source*:** refere-se ao componente origem do evento
    - ***event object*:** encapsula as informações sobre o evento ocorrido
    - ***event listener*:** quem monitora e responde ao evento

Java - GUI

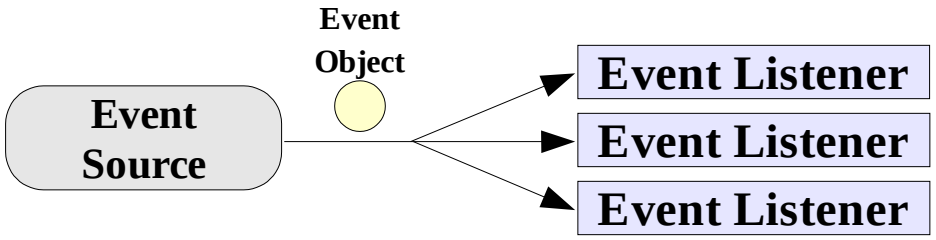


Fig. 1: Um *Event Source* pode estar associado a diversos *Event Listener*

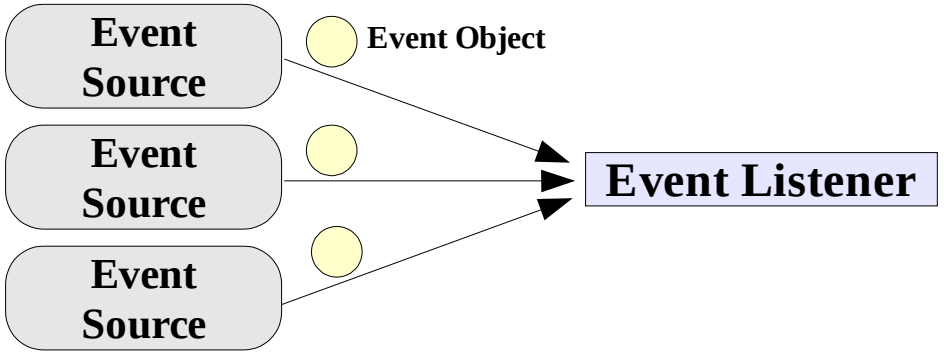


Fig. 2: Vários *Event Source* podem estar associados a um único *Event Listener*

Java - GUI

- Objetos de Evento (Event Object)
  - Todo método de um "Listener" tem um parâmetro que descende de EventObject
  - Exemplo: *ActionListener* possui o método *actionPerformed* cujo parâmetro é ***ActionEvent***.

```
public void actionPerformed (ActionEvent e) { ... }
```

  - Este objeto possuirá todas as informações necessárias sobre o evento.
  - Um dos métodos mais importantes é o getSource() que retorna qual objeto disparou o evento.

```
Object getSource();
```

Java - GUI

- Tratamento dos eventos
  - São executados em uma única thread - *event-dispatching thread*, com isso temos as seguintes implicações:

    1) Garante que cada executor terminará de atender uma requisição antes de atender a próxima.

    2) A interface não pode ser modificada enquanto o executor estiver processando uma ação de um evento (ex: código de renderização, cliques do mouse, ...).

    3) O código para atender os eventos deve ser pequeno e rápido, caso contrário recomenda-se criar novas *threads* para atender a requisição.

Java - GUI

- Exemplos de eventos e seus correspondentes *Listeners*
  - Clicar em um botão, menu ou pressionar Enter --> ***ActionListener***
  - Fechar um Frame --> ***WindowListener***
  - Pressionar o botão do mouse --> ***MouseListener ****
  - Mover o mouse sobre um componente --> ***MouseMotionListener***
  - Componente torna-se visível --> ***ComponentListener ****
  - Componente recebe o foco --> ***FocusListener ****
  - Seleção em tabela ou lista --> ***ListSelectionListener***
  - Pressionar uma tecla --> ***KeyListener ****
  - Alterar a posição do cursor em um texto --> ***CaretListener***
  - Mudança do estado da janela --> ***WindowStateListener***

(*) Todos os componentes swing implementam essas interfaces

Java - GUI

- Implementação - "Executor de eventos" (Event Handler)
  - Declaração da classe executora do evento

```
public class MyClass implements ActionListener { ...
```

  - Código que registra uma instância do executor de eventos

```
componente.addActionListener(instância_de_MyClass);
```

  - Código que implementa os métodos da interface

```
public void actionPerformed(ActionEvent e) {
   ...//código refente a ação
} //actionPerfomed
```

Java - GUI

- Classes Adaptadoras e Internas para tratar eventos (Adapters and Inner Class)
  - Maioria das interfaces contém mais que um método
  - Sempre que você for implementar apenas um método de uma interface, deve-se implementar todos os outros.
  - Exemplo *MouseListener:* mousePressed, mouseReleased, mouseEntered, mouseExited, and mouseClicked.
  - Solução: Usar classes adaptadoras e internas

Java - GUI

- Adapter Class
  - Implementa uma interface e fornece uma implementação *default* para os seus métodos, assim é necessário somente redefinir o método que será utilizado.

```
public class MyClass extends MouseAdapter {
   ...
someObject.addMouseListener(this);
   ...
   public void mouseClicked(MouseEvent e) {
        .....//implementação do método
   }
}
```

Java - GUI

- Inner Class
  - São classes embutidas dentro de outras classes. Geralmente são utilizadas como anônimas.

```
public class MyClass extends Applet {     ...
    someObject.addMouseListener(new MyAdapter());
    ...
    // Inner Class
    class MyAdapter extends MouseAdapter {
        public void mouseClicked(MouseEvent e) {
            ...// Implementação
        }
    } //class MyAdapter
} // class MyClass
```

```
// Anonymous Inner Class
someObject.addMouseListener(new MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            ...//Implementação
        }
});
```

Java - GUI

- Eventos - Interface ActionListener
  - void key actionPerformed (ActionEvent)
    - Processado após a execução de uma ação pelo usuário.
    - Exemplo: Clique do mouse, ENTER, selecionar opção do menu, ...

- Eventos – Interface KeyListener
  - void keyTyped(KeyEvent)
    - Processado após o usuário digitar um caracter.
  - void keyPressed(KeyEvent)
    - Processado após o usuário precionar uma tecla.
  - void keyReleased(KeyEvent)
    - Processado após o usuário soltar uma tecla.

Java - GUI

- Eventos - Interface MouseListener
  - void mouseClicked(MouseEvent)
    - Processado após o usuário clicar em um componente.
  - void mouseEntered(MouseEvent)
    - Processado após o cursor entrar nos limites do componente
  - void mouseExited(MouseEvent)
    - Processado após o cursor sair dos limites do componente
  - void mousePressed(MouseEvent)
    - Processado após o usuário pressionar o botão do mouse
  - void mouseReleased(MouseEvent)
    - Processado após o usuário soltar o botão do mouse

Java - GUI

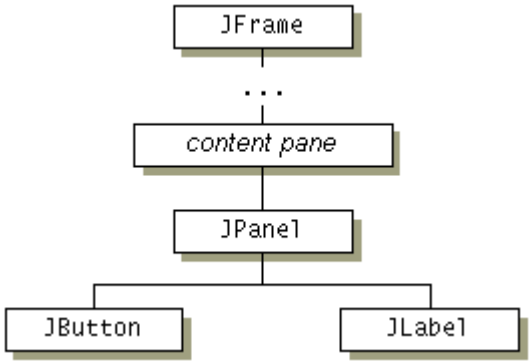
- Renderização dos Componentes (Painting)
  - Auto-renderização: Pode ocorrer na apresentação, quando o componente torna-se visível, ou na mudança de estado, quando esta tem influência sobre a GUI.
  - O processo ocorre das hierarquia de mais alto nível (containtment hierarchy) e segue até alcançar os componentes dos níveis inferiores.
  - O processo é gerenciado pelo *AWT Painting System,* contudo torna-se mais eficiente e suavizado pelo *gerenciador de pintura do Swing*.
  - Os componentes Swing se auto-renderizam sempre que necessário.

Java – GUI

- Renderização dos Componentes (Painting)
  - Programas devem ser renderizados somente quando o sistema pedir para eles, pois esta operação deve ser executada sem interrupção.
  - Para suavização, os componentes swing utilizam um buffer duplo – um para renderizar escondido do usuário (offscreen) e após o término enviar para o outro buffer, o qual será apresentado (onscreen).

Java - GUI

- Exemplo de Renderização



Passos executados:

1. JFrame (top-level container) se auto-renderiza
2. Content Pane pinta o fundo (retângulo cinza)
3. JPanel pinta seu fundo (retângulo cinza), depois a sua borda e por enfim pede para seus "filhos" se renderizarem.
4. JButton pinta seu fundo, o texto, o foco (se possuir)
5. JLabel pinta o seu texto.

Exercícios

- Projetar a interface abaixo, utilizando o **layout null**.

Exercícios

- Projetar a interface anterior utilizando a combinação dos gerenciadores de layout estudados: FlowLayout, BorderLayout, GridLayout, BoxLayout, GridBagLayout.
- Sugestões:
  - Utilizar containers do tipo JPanel
  - Utilizar GridBagLayout ou FlowLayout para "dados"
  - Utilizar BorderLayout com BoxLayout para "interesses"
  - Utilizar FlowLayout para "observações"
  - Utilizar BoxLayout ou GridLayout para os "botões"

Exercícios

- Implementar as funcionalidades da GUI do exercício anterior utilizando eventos e arquivos.

- Construir uma aplicação que utilize os eventos de clique do mouse e teclado. Sugestão: Um programa para desenhos de figuras vetoriais que com a tecla "Shift" pressionada desenha formas geométricas perfeitas como círculos, quadrados e triângulos equiláteros. O programa também deverá permitir ao usuário configurar as propriedades das figuras (cor de linha, cor de fundo, posicionamento) e fornecer opções para gravação e recuperação do conteúdo.

obs: quem fizer o programa acima terá a nota integral de todos os exercícios de swing.

Para saber mais

- Trail: Creating a GUI with JFC/Swing, http://java.sun.com/docs/books/tutorial/uiswing/index.html
- Lesson: Learning Swing by Example http://java.sun.com/docs/books/tutorial/uiswing/learn/index.html
- Deitel, H. M., Deitel, P. J. Java Como Programar. Bookman, quarta edição, 2003.