

Introdução à Programação



Interfaces Gráficas

Tópicos da Aula

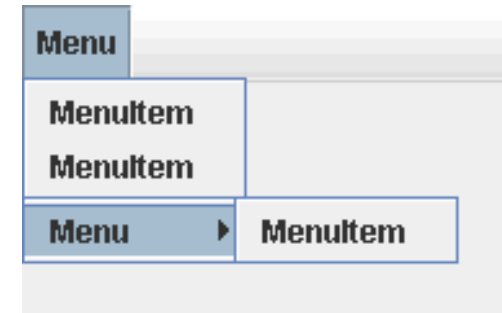
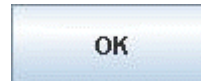
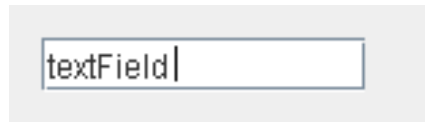
- ◆ Hoje vamos ver conceitos mais avançados de Interfaces Gráficas
 - Elementos de uma GUI
 - Componente
 - Eventos
 - Listeners
 - Padrão *Observer*
 - Painel
 - Exemplos com MiniJava
 - Exemplos com API gráfica de Java (Swing)

GUI

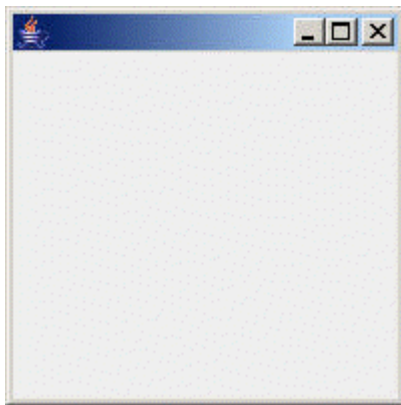
- ◆ Uma GUI em Java geralmente têm 3 tipos de objetos
 - Componentes
 - Eventos
 - Listeners
- ◆ Estes três tipos de objeto interagem para prover um comportamento dinâmico a interface gráfica

Componentes

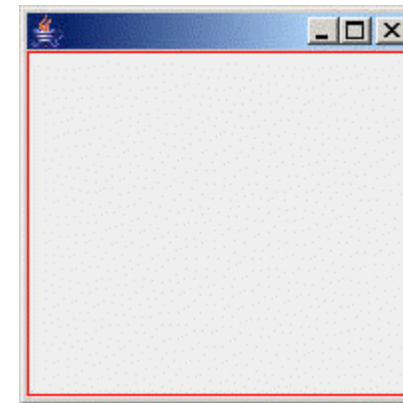
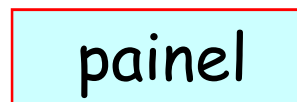
- ◆ Podem ser componentes elementares:



- ◆ Podem ser componentes que armazenam e organizam outros componentes (*containers*):



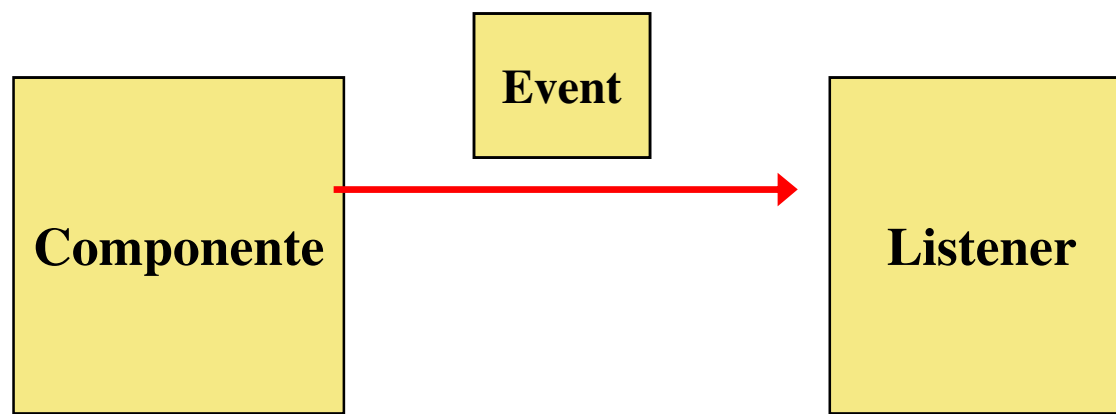
janela



Eventos e Listeners

- ◆ Um evento é um objeto que corresponde a um acontecimento que propicia uma ação em resposta
 - Clique ou movimento do mouse, pressionar um botão, pressionar uma tecla, etc
- ◆ Eventos, freqüentemente, correspondem a ações do usuário, porém nem sempre
- ◆ Listeners são objetos que esperam pelo acontecimento de um evento para responder com uma ação

Eventos e Listeners



Um componente pode
gerar um evento

Um listener é definido para
responder ao evento

Quando um evento ocorre, um componente chama
o método apropriado do listener,
passando o objeto que representa o evento

Desenvolvendo uma GUI

- ◆ Geralmente, utilizamos componentes e eventos predefinidos em classes de alguma biblioteca gráfica Java (ex: MiniJava)
- ◆ Portanto, geralmente, para criar programas em Java com GUI, devemos:
 - Instanciar e configurar componentes
 - Implementar listeners para os eventos desejados
 - Estabelecer as relações entre os listeners e componentes que geram os eventos

Padrão *Observer*

- ◆ Um ou mais objetos chamados de observadores (***observers***) se registram ou são registrados para monitorar eventos gerados pelo objeto observado (***subject***)
- ◆ Objeto Observado, geralmente, permite:
 - Adicionar ou remover *observers*
 - Notificar algum evento para os *observers*
- ◆ Objeto *Observer*, geralmente, permite:
 - Ser notificado
 - Executar uma ação em resposta ao evento

Padrão *Observer*

◆ Benefício:

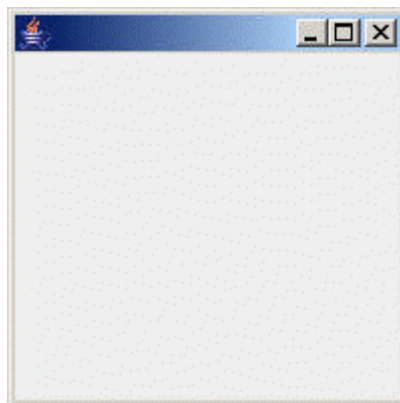
- Desacoplamento entre observado e observador
- Observado não precisa saber muito sobre observador, apenas permitir ser observado e repassar eventos

◆ GUIs em Java utilizam padrão *Observer*:

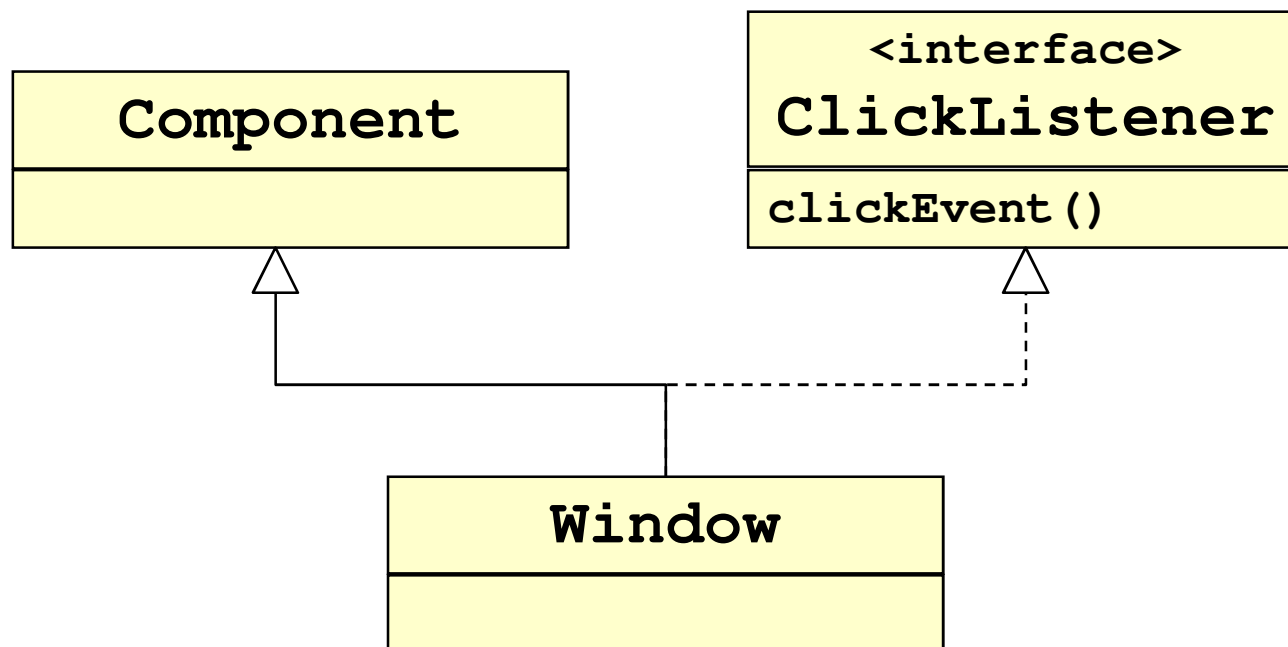
- Componentes são os observados
- Listeners são os observadores

Padrão *Observer* em MiniJava

Classe
Window



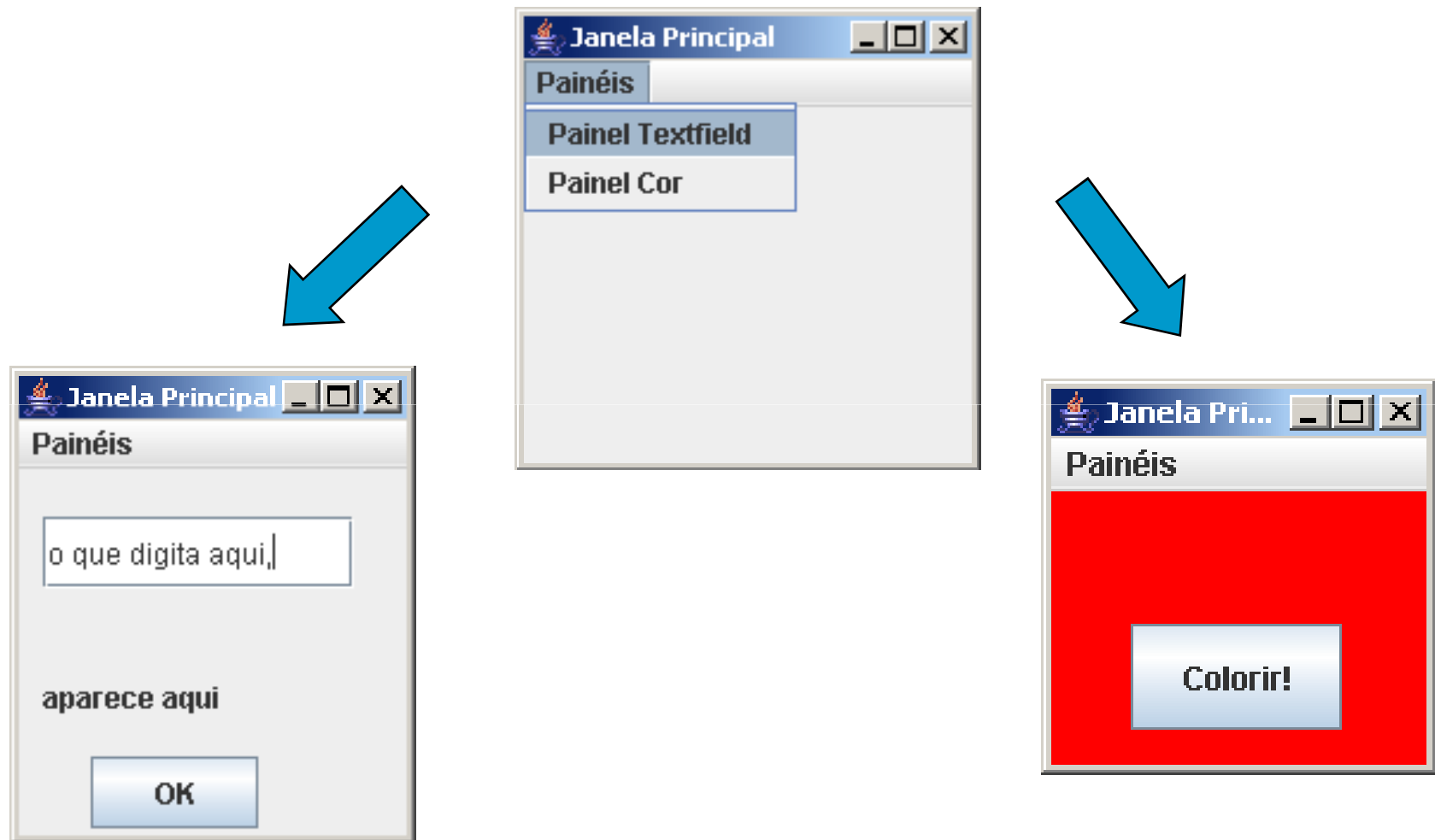
Containers são
observados e
observadores ao
mesmo tempo



Uso de *Panels*

- ◆ *Container*
 - Pode conter outros componentes
- ◆ Pequenas partes de uma janela
- ◆ Janela pode ser composta de múltiplos painéis
- ◆ Módulos independentes

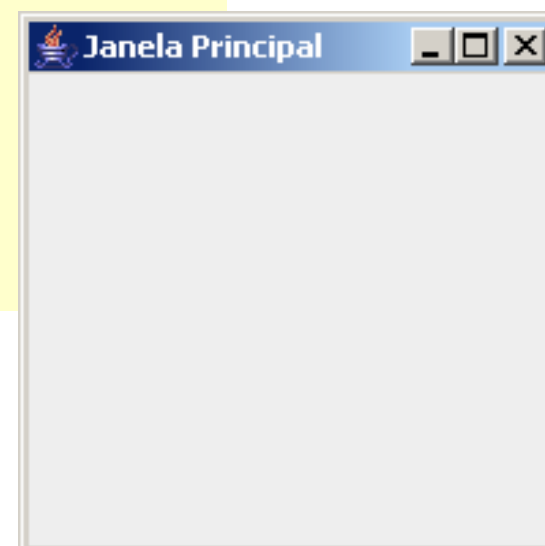
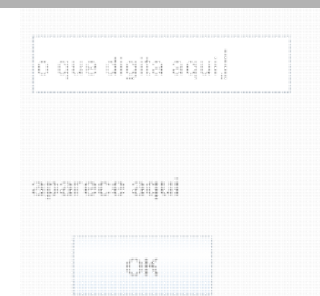
Panel - Exemplo



JanelaPrincipal.java

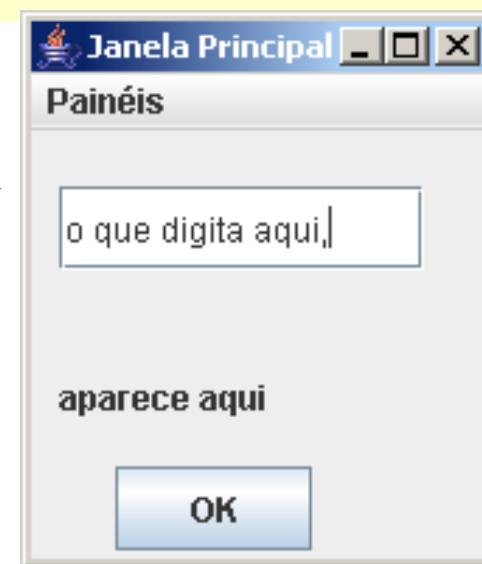
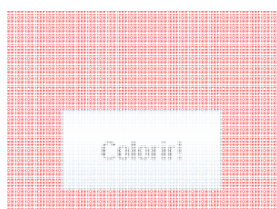
Panel - Exemplo

```
public JanelaPrincipal extends Window {  
    public JanelaPrincipal() {  
        super("Janela Principal");  
        pc = new PaineiCor();  
        pc.setVisible(false); this.include(pc);  
        ...  
        pt = new PaineiTextField();  
        pt.setVisible(false);  
        this.include(pt);  
        ...  
    }  
}
```



Panel - Exemplo

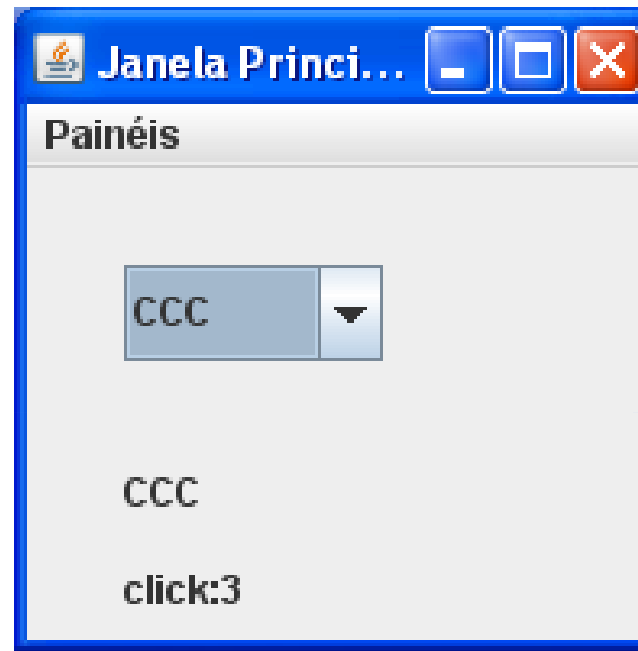
```
public void clickEvent (Component e) {  
    if (e == menuTextField) {  
        pc.setVisible(false);  
        pt.setVisible(true);  
        this.setSize (pt.getWidth(),  
                    pt.getHeight());  
    } ...  
}
```



Mais Listeners

- ◆ Componentes não precisam implementar comportamento de listeners
- ◆ Comportamento não se limita a ficar dentro da classe
 - *ClickListener*
 - *ItemStateListener*
- ◆ Eventos definidos para cada componentes
 - Não somente da janela inteira
- ◆ Vários listeners (ex: ComboBox)
 - *Veja PainelCombo.java*

Classe PanelCombo



Classe ComboStateAdapter

```
class ComboStateAdapter implements ItemStateListener{  
    private PainelCombo cmb;  
  
    public ComboStateAdapter(PainelCombo p){  
        cmb = p;  
    }  
  
    public void stateEvent(Component e){  
        cmb.acaoState();  
    }  
    public void stateEvent(){  
  
    }  
  
}
```

Ligando o
observador com o
observado

Classe ComboClickAdapter

```
class ComboClickAdapter implements ClickListener{  
    private PainelCombo cmb;  
  
    public ComboClickAdapter(PainelCombo p){  
        cmb = p;  
    }  
  
    public void clickEvent(Component e){  
        cmb.acaoClick();  
    }  
    public void clickEventEvent(){  
  
    }  
  
}
```

Ligando o
observador com o
observado

Classe PainelCombo

```
public class PainelCombo extends Panel{

    private ComboBox combo;
    private Label label1;
    private Label label2;
    private int contadorClick;
    public PainelCombo(){
        combo = new ComboBox();
        combo.addItem("AAA");
        combo.addItem("BBB");
        combo.addClickListener(new ComboClickAdapter(this));
        combo.addItemStateListener(new ComboStateAdapter(this));
        ...
    }

    public void acaoState(){
        label1.setText(combo.getSelectedItem().toString());
    }

    public void acaoClick(){
        contadorClick++;
        label2.setText("click:"+contadorClick);
    }
}
```

Registrando os
listeners

Comportamentos
serão invocados
pelos listeners

Pra quem quiser mais...

◆ javax.swing

- Mais recursos
 - Mais classes
 - Maior customização
- Disponível na API Java
- Eventos: análogo a eventos em MiniJava (*ActionListener*)

Exemplo de Janela com Múltiplos Painéis com Swing

```
public class JanelaPrincipalComSwing extends JFrame implements  
    ActionListener {
```

```
    private PainelCorComSwing pc;  
    private PainelTextfieldComSwing pt;  
    private JMenuBar barra;  
    private JMenu menu;  
    private JMenuItem menuItemTextfield;  
    private JMenuItem menuItemCor;  
    public JanelaPrincipalComSwing() {  
        pc = new PainelCorComSwing();  
        pc.setLocation(0,20);  
        this.add(pc);  
        barra = new JMenuBar();  
        menu = new JMenu("Painéis");  
        menuItemCor = new JMenuItem("Painel Cor");  
        menuItemCor.addActionListener(this);  
        this.setJMenuBar(barra);  
        barra.add(menu);  
        menu.add(menuItemCor);  
        this.setVisible(true);  
    }
```

Parecido com
Window de
MiniJava, mas não
implementa
listener

Listener para
eventos de ação

menuItemCor
registra esta
janela como
listener

Exemplo de Janela com Múltiplos Painéis com Swing

@Override

```
public void actionPerformed(ActionEvent e) {  
  
    if(e.getSource() == menuTextField){  
        pc.setVisible(false);  
        pt.setVisible(true);  
        this.setSize(pt.getWidth(), pt.getHeight());  
    }else if(e.getSource() == menuCor){  
        pc.setVisible(true);  
        pt.setVisible(false);  
        this.setSize(pc.getWidth(), pc.getHeight());  
    }  
}
```

Este método deve
ser implementado
para o
ActionListener

Método que
retorna o
componente que
gerou o evento

Referências

◆ MiniJava

- <http://www.cin.ufpe.br/~if669/index.php/MiniJava>

◆ Java

- <http://java.sun.com/reference/api/>

Resumindo

- ◆ Elementos de uma GUI
 - Componente
 - Eventos
 - Listeners
- ◆ Interação entre elementos de uma GUI
- ◆ Padrão *Observer*
- ◆ Painel
- ◆ Separando o listener do componente
- ◆ Exemplos com MiniJava
- ◆ Exemplos com API gráfica de Java (Swing)