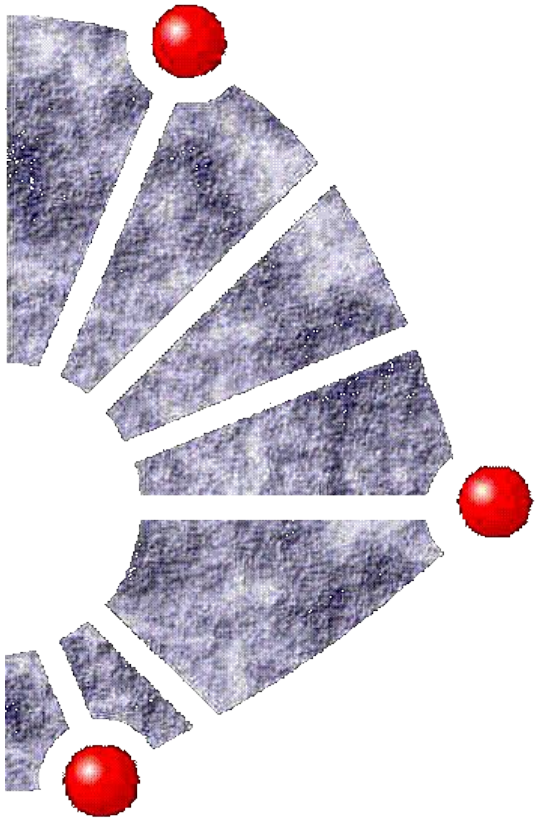
Implementando Interfaces Gráficas: AWT e Swing



AULA 2

Interfaces Gráficas com o Usuário

- Interfaces Gráficas com o Usuário (GUI)
  - parte fundamental da maioria dos programas em Java
- Widgets
  - componentes de uma interface gráfica: *textboxes, listboxes, radioboxes, buttons, canvases, menus e submenus, sliders*, etc..
- Cada Pacote Gráfico ou S.O. Gráfico
  - possui um conjunto próprio de widgets
    - quanto a funcionalidade
    - quanto a apresentação visual
  - existe um conjunto mínimo que corresponde à interseção de todos os pacotes gráficos - AWT (Abstract Window Toolkit)
- Java e GUIs
  - Java Foundation Classes

JFC - Java Foundation Classes

- Origem do JFC
  - Junto com o Java 1.0, a Sun oferece o AWT
  - AWT se mostra insuficiente para o desenvolvimento de aplicações mais sofisticadas
  - Netscape aparece com o IFC - Internet Foundation Classes
  - Microsoft aparece com o AFC - Application Foundation Classes
  - Sun/JavaSoft, IBM e Netscape desenvolvem então o JFC
- JFC
  - estendem o AWT por meio da inclusão de diversos novos *features* que já haviam aparecido no IFC e AFC, além de diversas outras inovações
  - passa a incorporar a API oficial do Java, evitando a necessidade de downloads de classes especiais a cada caso

JFC - Java Foundation Classes

- Componentes do JFC
  - AWT
    - Toolkit de Janelas Abstrato
  - Java™ 2D
    - API gráfica bidimensional baseada em uma tecnologia licensiada da IBM/Taligent
  - Accessibility
    - provê tecnologias assistivas, tais como magnificadores de tela, para o uso em diversas partes do JFC
  - Drag and Drop
    - parte do modelo JavaBeans, disponível a partir da plataforma Java 2 (Java 1.2)
  - Swing
    - conjunto de widgets formando um framework que especifica uma interface gráfica independente de plataforma

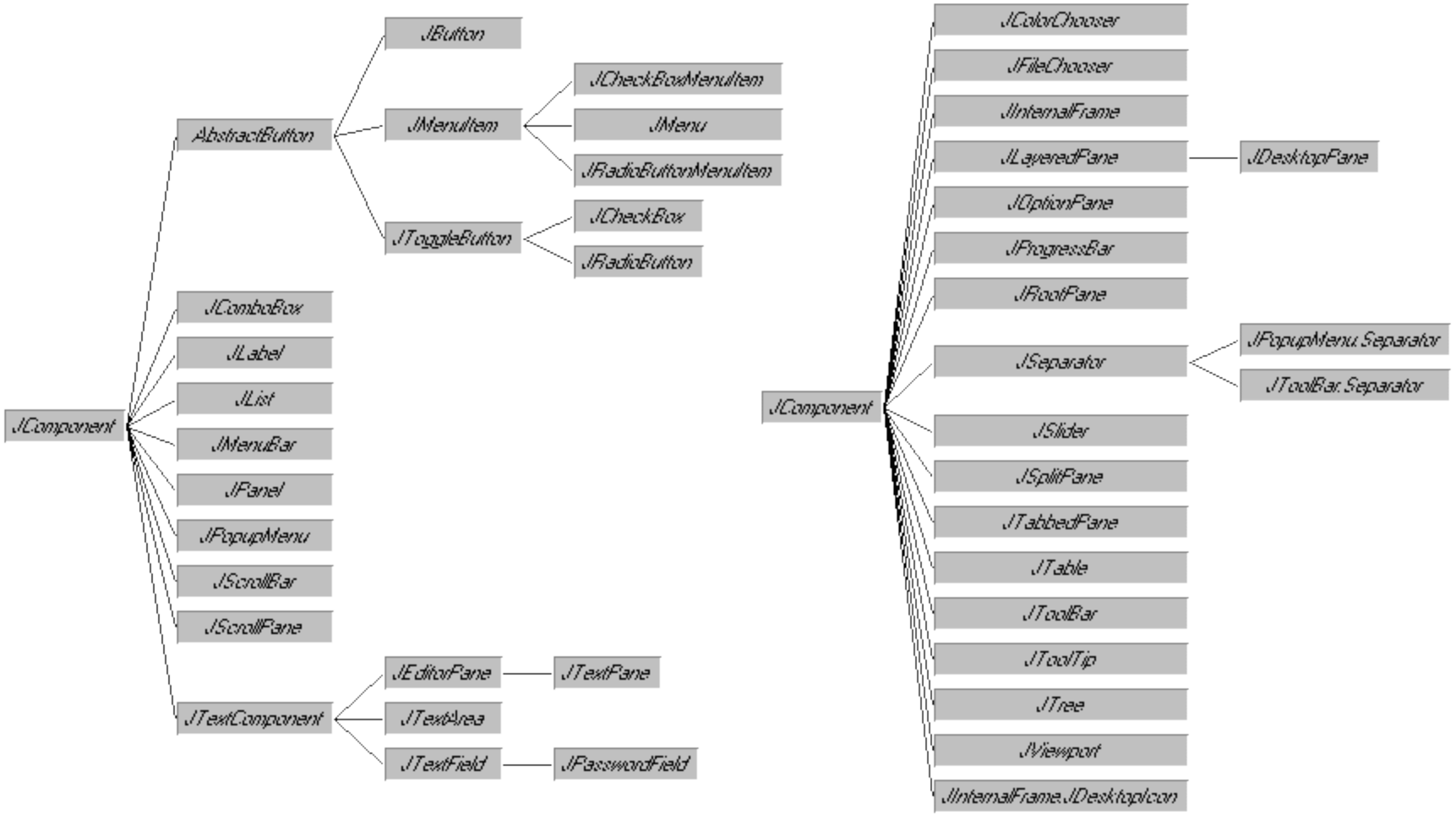
AWT - Abstract Window Toolkit

- Toolkit de Janelas Abstrato
  - fornece um conjunto de classes *wrapper* que permite a utilização de *widgets* de diferentes pacotes gráficos em diferentes plataformas - abstração dos modelos de *widgets*
- Problemas
  - não uniformidade de apresentação visual das *widgets*, em diferentes plataformas
  - não aproveitamento (impossibilidade de utilização) de características avançadas dos widgets nativos
  - conjunto-interseção de *widgets* disponíveis em todas as plataformas é restrito demais para aplicações sofisticadas
- Recomendação de Uso
  - somente para Applets ou Aplicações muito simples

Swing

- *Widgets* do Swing
  - estendem o AWT, por meio da adição de um novo componente base, o JComponent, derivando uma nova hierarquia de classes de interface, que aderem ao modelo JavaBeans e seu modelo de eventos
- Subconjunto das *Widgets* do Swing
  - análogos aos *widgets* básicos do AWT
  - componentes *lightweight*, com *look-and-feel* plugável
- Arquitetura de Componentes *Lightweight*
  - introduzida no AWT 1.1
  - permite a existência de componentes independentes do S.O.
  - Utilizam a arquitetura MVC (Model/View/Controller)
- Novos Widgets
  - Trees, Tabbed Panes e Splitter Panes

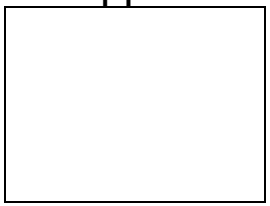
Widgets do Swing



Widgets do Swing

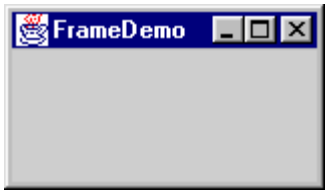
Top Level Containers

Applet



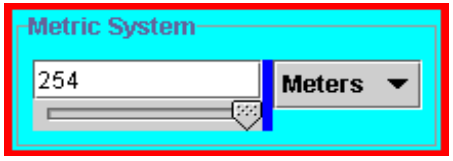
Dialog

Frame



General Purpose Containers

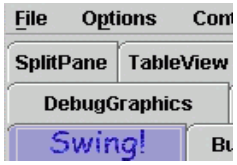
Panel



Scroll Pane

Split Pane

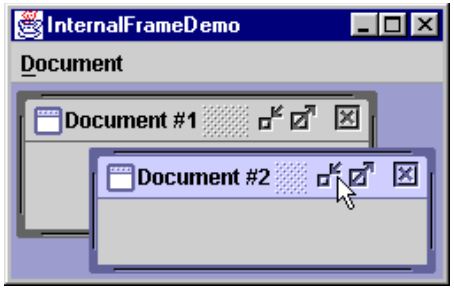
Tabbed Pane



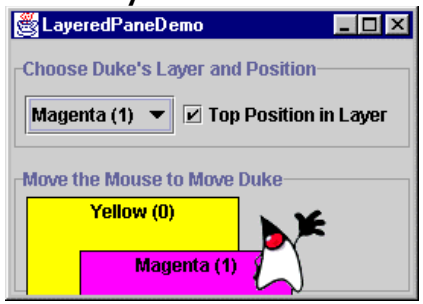
Tool Bar

Special Purpose Containers

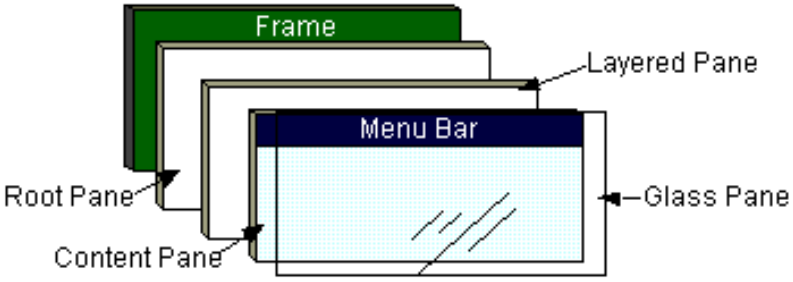
Internal Frame



Layered Pane



Root Pane

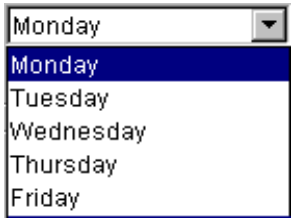


Widgets do Swing

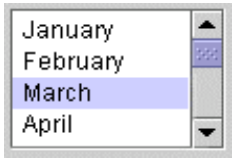
Basic Controls

Buttons

Combo Box



List



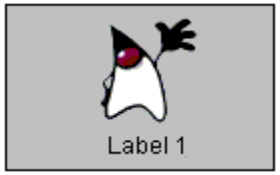
Menu

Slider

Text

Uneditable Information Displays

Label

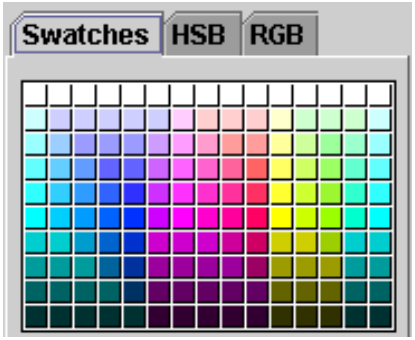


Progress Bar

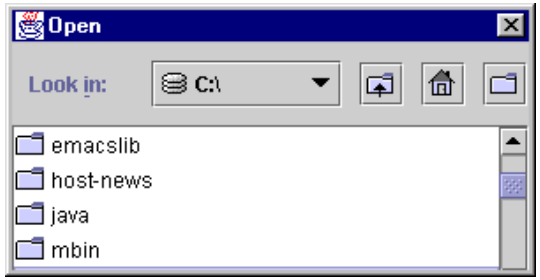
Tool Tip

Editable Displays of Formatted Information

Color Chooser

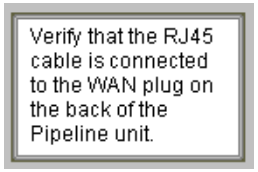


File Chooser



Table

Text



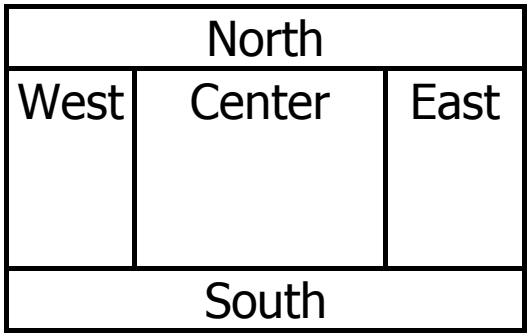
Tree

Criando e Gerenciando Janelas

- Painéis
  - JPanel - o *container* básico
  - área retangular que pode conter outras widgets, ser desenhada e capturar eventos
  - caso um JPanel contenha outras widgets (inclusive outros JPanels, recursivamente), ele deve usar um LayoutManager
- LayoutManager
  - objeto que organiza as *widgets* dentro de um *container*
  - BorderLayout, GridLayout, FlowLayout, etc ...
  - LayoutManager default é o BorderLayout
- A Janela Principal
  - JFrame - uma janela com decorações
  - possui um JPanel (chamado de painel de conteúdo)

Criando e Gerenciando Janelas

- Visibilidade
  - JFrames são criados normalmente invisíveis, podendo ser tornados visíveis por meio do método setVisible()
  - Outros tipos de widgets normalmente são criados visíveis
  - Tornar uma widget invisível não a destrói
- O LayoutManager BorderLayout
  - distribui widgets segundo o mapa ao lado
  - Os termos North, South, East, West e Center são chamados de constraints do LayoutManager
  - Podem ser acessados por meio de variáveis estáticas e especificados durante a inserção de novos JComponents a um Jpanel
  - add(button1, BorderLayout.CENTER);



Modelo de Eventos

- Modelo de Eventos do Swing
  - Dispositivos de Entrada (Teclado, Mouse, etc) geram eventos
  - Janelas interessadas em receber eventos devem indicar seu interesse registrando um *Listener* apropriado ao evento em questão
- *Listeners*
  - objetos que implementam uma interface derivada da interface EventListener
    - ActionListener, ChangeListener, FocusListener, KeyListener, MouseListener, MouseMotionListener, WindowListener, etc...
  - Cada tipo de EventListener é responsável por um determinado tipo de eventos
  - *Widgets* aceitam *Listeners* condizentes com sua funcionalidade
  - Para que uma *widget* recebe eventos, um *Listener* apropriado deve ser adicionado à widget

Modelo de Eventos

- Implementação de um *Listener*
  - implica na definição de todos os métodos declarados no *Listener*
  - em alguns casos, podem implicar em trabalho desnecessário
- Adapters
  - Conveniência para a criação de objetos do tipo Listener
  - Classes implementando alguns tipos de *Listener* mais comuns, com implementações vazias
  - desenhados para serem estendidos por classes interessadas, evitando a declaração de métodos que não serão utilizados
  - métodos utilizados sofrem *override*
- Adapters Disponíveis
  - ComponentAdapter, ContainerAdapter, FocusAdapter, KeyAdapter, MouseAdapter, MouseMotionAdapter, WindowAdapter, etc...

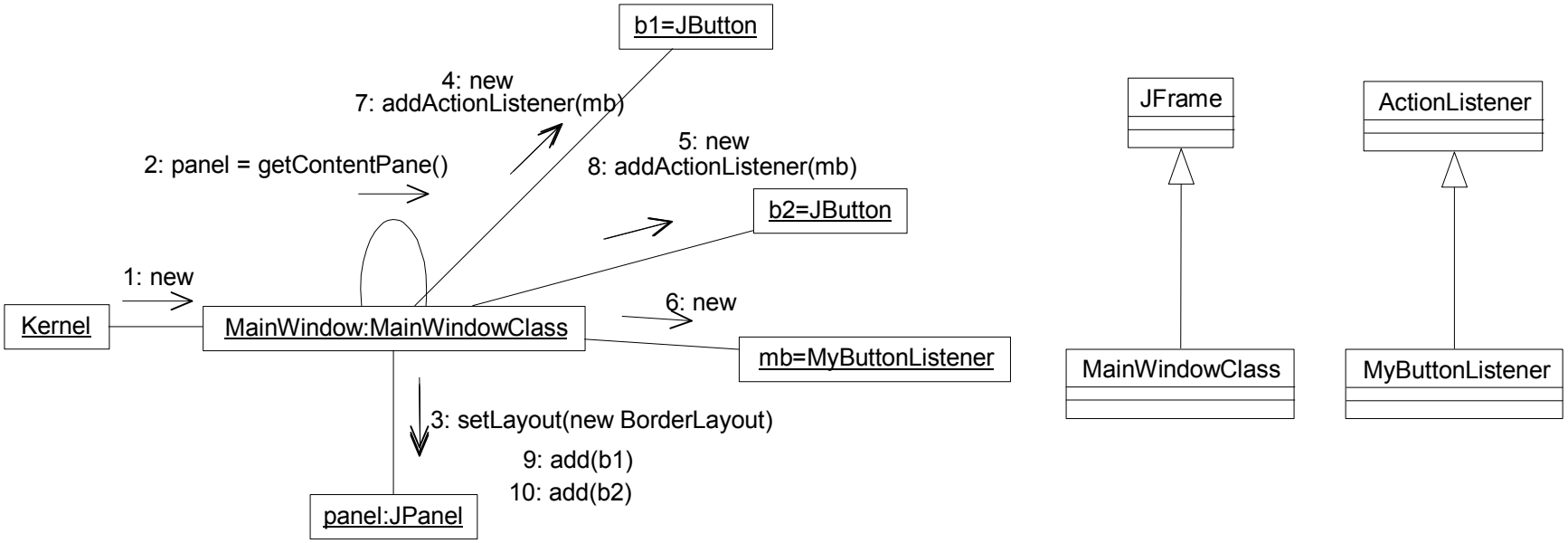
Modelo de Eventos

- Exemplo
  - MouseListener
    - `void mouseClicked(MouseEvent e)`
    - `void mouseEntered(MouseEvent e)`
    - `void mouseExited(MouseEvent e)`
    - `void mousePressed(MouseEvent e)`
    - `void mouseReleased(MouseEvent e)`
  - MouseAdapter
    - implementa funções para cada um dos protótipos acima:
    - `void mouseClicked(MouseEvent e) { }`
    - `void mouseEntered(MouseEvent e) { }`
    - `void mouseExited(MouseEvent e) { }`
    - `void mousePressed(MouseEvent e) { }`
    - `void mouseReleased(MouseEvent e) { }`

Modelo de Eventos

- Dois Mecanismos p/ Eventos: Listeners e Adapters
  - Listener - interface (deve ser implementada)
  - Adapter - classe (deve ser estendedida)
- Exemplos
  - p = new JPanel();
    - class MyListener implements MouseListener { ... }
    - class MyListener extends MouseAdapter { ... }
  - p.addMouseListener(new MyListener());
- Informações sobre o evento
  - podem ser obtidas invocando métodos ao próprio evento:
    - void mouseClicked(MouseEvent e) {
    - int x,y;
    - x = e.getX(); y = e.getY(); ... }

Exemplo de Criação de uma Janela com 2 Botões



Diferentes Maneiras de Implementar um Listener

- Uso de Classes Adapters Separadas

```
Class MyAdapter implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        ...
    }
}

Class MyPanel extends JPanel {
    void MyPanel() {
        ...
        JButton b1 = new JButton();
        b1.addActionListener(new MyAdapter());
        ...
    }
}
```

Diferentes Maneiras de Implementar um Listener

- Própria Classe implementa o Listener

```
Class MyPanel extends JPanel implements ActionListener {
    void MyPanel() {
        ...
        JButton b1 = new JButton();
        b1.addActionListener(this);
        ...
    }
    public void actionPerformed(ActionEvent e) {
    ...
    }
}
```

Diferentes Maneiras de Implementar um Listener

- Inner Classes anônimas

```
Class MyPanel extends JPanel {
    void MyPanel() {
        ...
        JButton b1 = new JButton();
        b1.addActionListener(new ActionListener()  {
              public void actionPerformed(ActionEvent e) {
                 FunctionListener(e);
              }
         });
        ...
    }

    void FunctionListener(ActionEvent e) {
    ...
    }
}
```

Desenhando em uma Janela

- Todos os JComponents podem ser desenhados
  - como: fazendo um *override* do método "paint"
  - `public void paint(Graphics g)`
- O objeto "Graphics"
  - provê um conjunto de métodos para desenho, *filling*, etc...
- Graphics2D
  - estende a classe Graphics de modo a proporcionar um melhor controle sobre a geometria, transformação de coordenadas, gerenciamento de cores e layout de texto
  - classe fundamental para o desenho de formas bidimensionais, texto e imagens
  - ```
    public void paint(Graphics g)
        {Graphics2D g2 = (Graphics2D) g;
         …
        }
    ```
    ```

Desenhando em uma Janela

- Exemplos de Métodos em Graphics2D
  - fill, fillRect, fillRoundRect, fillOval, fillArc, fillPolygon, and clearRect
  - draw, drawLine, drawRect, drawRoundRect, drawOval, drawArc, drawPolyline, and drawPolygon, drawImage, drawString
  - clip, setClip, setColor, setFont setPaintMode, setXORMode
- A Interface Shape
  - interface implementada pelas classes:
    - Polygon, RectangularShape, Rectangle, Area, Line2D, GeneralPath, QuadCurve2D, CubicCurve2D
  - objetos que implementam a interface Shape podem ser passados como parâmetros para uma série de métodos de desenho, tais como fill, draw, clip, etc ...