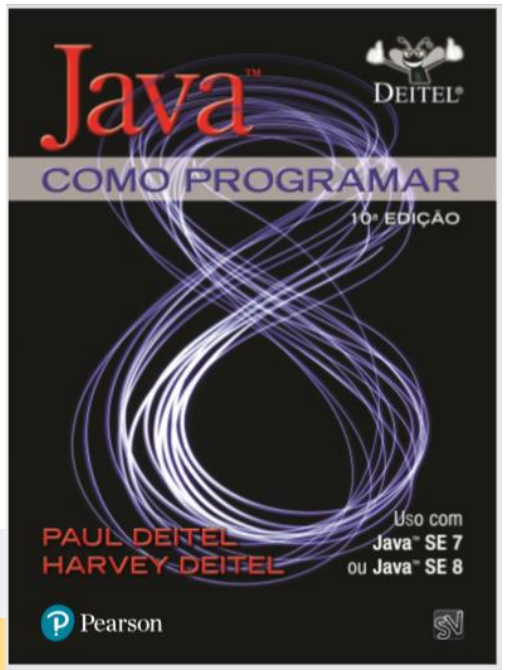
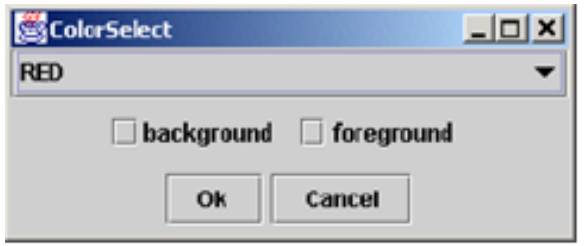


Java – Aula 7
Janelas e Menus

Notas de Aula
Prof. André Bernardi
andrebernardi@unifei.edu.br

Janelas em Java - JFrame



- Um ***JFrame*** é uma janela com uma barra de título e uma borda.

- A classe ***JFrame*** é uma subclasse de ***Frame*** (pacote java.awt), que é uma subclasse de ***Window*** (pacote java.awt).

- Assim, ***JFrame*** é um dos componentes GUI Swing pesados.

Exibindo e posicionando janelas

- Ao exibir uma janela em um programa Java, a janela é fornecida pelo conjunto de ferramentas de janelas da plataforma local e, portanto, ela irá parecer qualquer outra janela exibida nessa plataforma.

- Por padrão, uma janela só é exibida na tela depois que o programa invoca o método ***setVisible*** da janela (herdado da classe java.awt.Component) com um argumento *true*.

Exibindo e posicionando janelas

- O tamanho de uma janela deve ser configurado com uma chamada ao método ***setSize*** (herdado da classe java.awt.Component).

- A posição de uma janela, quando aparece, na tela é especificada com o método ***setLocation*** (herdado da classe java.awt.Component)

- Eventos são gerados quando o usuário manipula a janela.
- Um ***WindowListener*** é registrado para eventos de janela com o método ***addWindowListener***.
- A interface ***WindowListener*** fornece ***sete*** métodos de tratamento de eventos de janela —
  - ***windowActivated*** - chamado quando o usuário torna uma janela a janela ativa,
  - ***windowClosed*** - chamado depois de a janela ser fechada,
  - ***windowClosing*** - chamado quando o usuário inicia o fechamento da janela,
  - ***windowDeactivated*** - chamado quando o usuário torna outra janela a janela ativa,
  - ***windowDeiconified*** - chamado quando o usuário restaura uma janela que está minimizada,
  - ***windowIconified*** - chamado quando o usuário minimiza uma janela,
  - ***windowOpened*** - chamado quando um programa exibe uma janela na tela pela primeira vez

Usando Menus em Janelas

- Classe **JMenuBar**
  - Método setJMenuBar da JFrame
- Classe **JMenu**
- Classe **JMenuItem**
  - **JCheckBoxMenuItem**
  - **JRadioButtonMenuItem**

Usando Menus em Janelas

- Menus organizam perfeitamente os comandos em uma GUI. Em *GUIs Swing*, os menus só podem ser anexados a objetos das classes com o método ***setJMenuBar***.

- Um ***JMenuBar*** é um contêiner para menus. Um ***JMenuItem*** aparece em um menu. Um ***JMenu*** contém itens de menu e pode ser adicionado a um ***JMenuBar*** ou a outros ***JMenus*** como submenus.

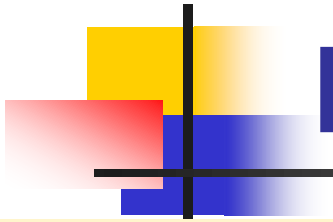
Usando Menus em Janelas

- Quando um menu é clicado, ele se expande para mostrar sua lista dos itens de menu.
- Os menus aparecem da esquerda para a direita na ordem em que eles são adicionados a um ***JMenuBar***.
- Quando um ***JCheckBoxMenuItem*** é selecionado, uma marca de verificação aparece à esquerda do item de menu. Quando o ***JCheckBoxMenuItem*** é selecionado novamente, a marca é removida.

Usando Menus em Janelas

- Em um ***ButtonGroup***, um único ***JRadioButtonMenuItem*** pode ser selecionado de cada vez.

- O método ***AbstractButton setMnemonic*** especifica o mnemônico para um botão. Os caracteres mnemônicos normalmente são exibidos com um caractere de sublinhado.



Exemplo

- Executar o exemplo MenuTest :
  - Observar a criação da JMenuBar e sua adição a janela do aplicativo;
  - Observar a criação dos Objetos JMenu;
  - Observar a criação e adição de objetos JMenuItem a objetos JMenu;
  - Observar o uso do método *setMnemonic*(‘’);
  - Notar que a sequência do menu é a mesma da de inserção dos objetos na JMenuBar.

```
1 // Figura 22.5: MenuFrame.java
2 // Demonstrando menus.
3 import java.awt.Color;
4 import java.awt.Font;
5 import java.awt.BorderLayout;
6 import java.awt.event.ActionListener;
7 import java.awt.event.ActionEvent;
8 import java.awt.event.ItemListener;
9 import java.awt.event.ItemEvent;
10 import javax.swing.JFrame;
11 import javax.swing.JRadioButtonMenuItem;
12 import javax.swing.JCheckBoxMenuItem;
13 import javax.swing.JOptionPane;
14 import javax.swing.JLabel;
15 import javax.swing.SwingConstants;
16 import javax.swing.ButtonGroup;
17 import javax.swing.JMenu;
18 import javax.swing.JMenuItem;
19 import javax.swing.JMenuBar;
20
21 public class MenuFrame extends JFrame
22 {
23    private final Color[] colorValues =
24       {Color.BLACK, Color.BLUE, Color.RED, Color.GREEN};
```

```
25  private final JRadioButtonMenuItem[] colorItems; // itens do menu Color
26  private final JRadioButtonMenuItem[] fonts;     // itens do menu Font
27  private final JCheckBoxMenuItem[] styleItems;   // itens do menu Font Style
28  private final JLabel displayJLabel;             // exibe texto de exemplo
29  private final ButtonGroup fontButtonGroup;      // gerencia itens do menu Font
30  private final ButtonGroup colorButtonGroup;     // gerencia itens do menu Color
31  private int style; // utilizado para criar estilos de fontes
32
33  // construtor sem argumento para configurar a GUI
34  public MenuFrame()
35  {
36      super("Using JMenus");
37
38      JMenu fileMenu = new JMenu("File");      // cria o menu File
39      fileMenu.setMnemonic('F');               // configura o mnemônico como F
40
41      // cria item de menu About...
42      JMenuItem aboutItem = new JMenuItem("About...");
43      aboutItem.setMnemonic('A');      // configura o mnemônico com A
44      fileMenu.add(aboutItem);         // adiciona o item about ao menu File
45      aboutItem.addActionListener(
46          new ActionListener()         // classe interna anônima
47          {
```

```
48              // exibe um diálogo de mensagem quando o usuário seleciona About...
49              @Override
50              public void actionPerformed(ActionEvent event)
51              {
52                  JOptionPane.showMessageDialog(MenuFrame.this,
53                      "This is an example\nof using menus",
54                      "About", JOptionPane.PLAIN_MESSAGE);
55              }
56          }
57      );
58
59      JMenuItem exitItem = new JMenuItem("Exit"); // cria o item exit
60      exitItem.setMnemonic('x');          // configura o mnemônico como x
61      fileMenu.add(exitItem);             // adiciona o item exit ao menu File
62      exitItem.addActionListener(
63          new ActionListener()            // classe interna anônima
64          {
65              // termina o aplicativo quando o usuário clica exitItem
66              @Override
67              public void actionPerformed(ActionEvent event)
68              {
69                  System.exit(0);         // encerra o aplicativo
70              }
71          }
72      );
```

```
74      JMenuBar bar = new JMenuBar();    // cria a barra de menus
75      setJMenuBar(bar);                 // adiciona uma barra de menus ao aplicativo
76      bar.add(fileMenu);                // adiciona o menu File à barra de menus
77
78      JMenu formatMenu = new JMenu("Format");         // cria o menu Format
79      formatMenu.setMnemonic('r');                    // configura o mnemônico como r
80
81      // array listando cores de string
82      String[] colors = { "Black", "Blue", "Red", "Green" };
83
84      JMenu colorMenu = new JMenu("Color");           // cria o menu Color
85      colorMenu.setMnemonic('C');                     // configura o mnemônico como C
86
87      // cria itens de menu de botão de rádio para cores
88      colorItems = new JRadioButtonMenuItem[colors.length];
89      colorButtonGroup = new ButtonGroup();           // gerencia cores
90      ItemHandler itemHandler = new ItemHandler();    // rotina de tratamento para cores
91                                                      //implementa uma ActionListener
92      // cria itens do menu Color com botões de opção
93      for (int count = 0; count < colors.length; count++)
94      {
95            colorItems[count] =
96                  new JRadioButtonMenuItem(colors[count]); // cria o item
```

```
97          colorMenu.add(colorItems[count]);   // adiciona o item ao menu Color
98          colorButtonGroup.add(colorItems[count]);    // adiciona ao grupo
99          colorItems[count].addActionListener(itemHandler);
100     }
101
102     colorItems[0].setSelected(true); // seleciona o primeiro item Color
103
104     formatMenu.add(colorMenu);      // adiciona o menu Color ao menu Format
105     formatMenu.addSeparator();      // adiciona um separador no menu
106
107     // array listando nomes de fonte
108     String[] fontNames = { "Serif", "Monospaced", "SansSerif" };
109     JMenu fontMenu = new JMenu("Font"); // cria a fonte do menu
110     fontMenu.setMnemonic('n');          // configura o mnemônico como n
111
112     // cria itens do menu radio button para nomes de fonte
113     fonts = new JRadioButtonMenuItem[fontNames.length];
114     fontButtonGroup = new ButtonGroup(); // gerencia os nomes das fontes
115
116     // criar itens do menu Font com botões de opção
117     for (int count = 0; count < fonts.length; count++)
118     {
119         fonts[count] = new JRadioButtonMenuItem(fontNames[count]);
120         fontMenu.add(fonts[count]);     // adiciona fonte ao menu Font
```

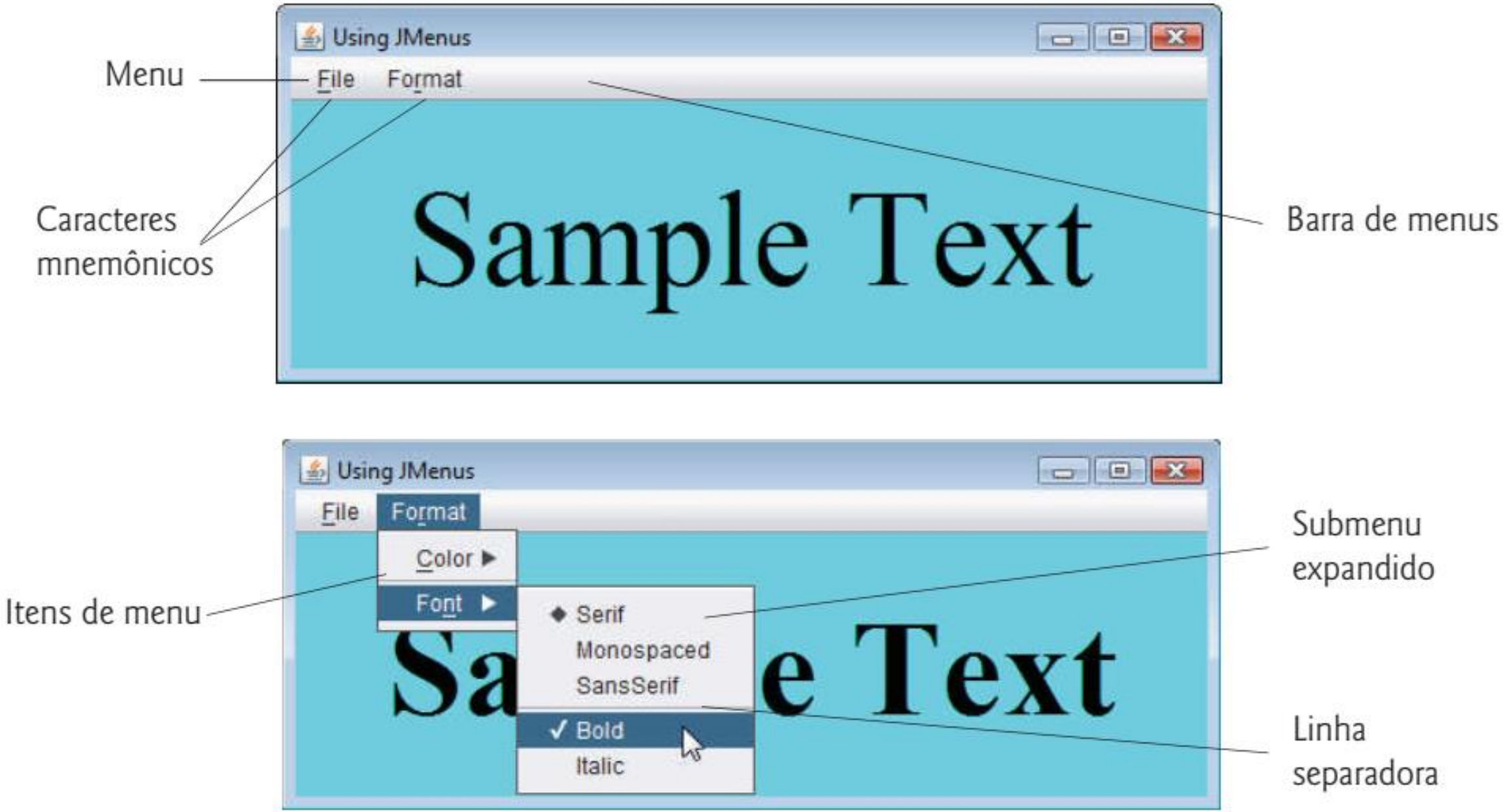
```
121         fontButtonGroup.add(fonts[count]); // adiciona ao grupo de botões
122         fonts[count].addActionListener(itemHandler); // registra evento
123      }
124
125      fonts[0].setSelected(true); // seleciona o primeiro item do menu Font
126      fontMenu.addSeparator(); // adiciona uma barra separadora ao menu Font
127
128      String[] styleNames = { "Bold", "Italic" }; // nomes dos estilos
129      styleItems = new JCheckBoxMenuItem[styleNames.length];
130      StyleHandler styleHandler = new StyleHandler(); // tratamento de estilo
131
132      // cria itens do menu Style com caixas de seleção
133      for (int count = 0; count < styleNames.length; count++)
134      {
135         styleItems[count] =
136          new JCheckBoxMenuItem(styleNames[count]); // para estilo
137         fontMenu.add(styleItems[count]); // adiciona ao menu Font
138         styleItems[count].addItemListener(styleHandler); // registrar
139      }
140
141      formatMenu.add(fontMenu); // adiciona o menu Font ao menu Format
142      bar.add(formatMenu); // adiciona o menu Format à barra de menus
143
```

```
144      // configura o rótulo para exibir texto
145      displayJLabel = new JLabel("Sample Text", SwingConstants.CENTER);
146      displayJLabel.setForeground(colorValues[0]);
147      displayJLabel.setFont(new Font("Serif", Font.PLAIN, 72));
148
149      getContentPane().setBackground(Color.CYAN); // configura o fundo
150      add(displayJLabel, BorderLayout.CENTER); // adiciona displayJLabel
151   } // fim do construtor de MenuFrame
152
153   // classe interna para tratar eventos de ação dos itens de menu
154   private class ItemHandler implements ActionListener
155   {
156      // processa seleções de cor e fonte
157      @Override
158      public void actionPerformed(ActionEvent event)
159      {
160         // processa a seleção de cor
161         for (int count = 0; count < colorItems.length; count++)
162         {
163            if (colorItems[count].isSelected())
164            {
165               displayJLabel.setForeground(colorValues[count]);
166               break;
167            }
168         }
```

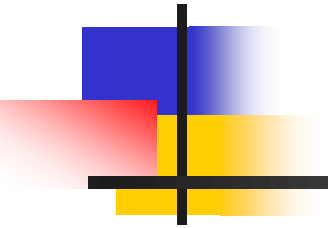
```
170         // processa a seleção de fonte
171         for (int count = 0; count < fonts.length; count++)
172         {
173            if (event.getSource() == fonts[count])
174            {
175               displayJLabel.setFont(
176                  new Font(fonts[count].getText(), style, 72));
177            }
178         }
179
180         repaint();      // redesenha o aplicativo
181      }
182   } // fim da classe ItemHandler
183
184   // classe interna para tratar eventos
185   private class StyleHandler implements ItemListener
186   {
187      // processa seleções de estilo da fonte
188      @Override
189      public void itemStateChanged(ItemEvent e)
190      {
191         String name = displayJLabel.getFont().getName(); // fonte atual
192         Font font; // nova fonte com base nas seleções pelo usuário
```

```
193
194         // determina quais itens estão marcados e cria Font
195         if (styleItems[0].isSelected() &&
196                 styleItems[1].isSelected())
197             font = new Font(name, Font.BOLD + Font.ITALIC, 72);
198             else if (styleItems[0].isSelected())
199                 font = new Font(name, Font.BOLD, 72);
200                 else if (styleItems[1].isSelected())
201                     font = new Font(name, Font.ITALIC, 72);
202                 else
203                 font = new Font(name, Font.PLAIN, 72);
204
205             displayJLabel.setFont(font);
206             repaint(); // redesenha o aplicativo
207         }
208     }
209 } // fim da classe MenuFrame
```

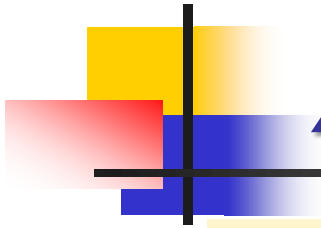
```
1 // Figura 22.6: MenuTest.java
2 // Testando MenuFrame.
3 import javax.swing.JFrame;
4
5 public class MenuTest
6 {
7     public static void main(String[] args)
8     {
9           MenuFrame menuFrame = new MenuFrame();
10          menuFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11          menuFrame.setSize(500, 200);
12          menuFrame.setVisible(true);
13    }
14 } // fim da classe MenuTest
```


Menu
Caracteres mnemônicos
Using JMenus
File
Format
Sample Text
Barra de menus
Using JMenus
File
Format
Color
Font
Serif
Monospaced
SansSerif
Bold
Italic
Sa e Text
Itens de menu
Submenu expandido
Linha separadora

Aparência e comportamento plugáveis



Look and Feel



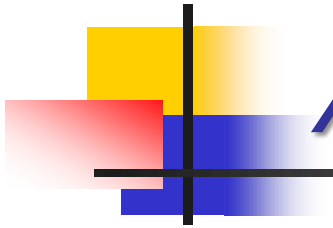
Aparência e comportamento plugáveis

- Componentes GUI muitas vezes têm uma aparência diferente em plataformas distintas (fontes, tamanhos de fonte, bordas de componente etc.) e podem exigir diferentes quantidades de espaço para que sejam exibidos.
- Isso poderia alterar os layouts e alinhamentos da GUI.

Aparência e comportamento plugáveis

- Componentes GUI em plataformas diferentes têm funcionalidades - padrão diferentes, por exemplo, nem todas as plataformas permitem que um botão com o foco seja "pressionado" com a barra de espaço.

- Componentes GUI de peso leve do Swing eliminam muitas dessas questões fornecendo funcionalidades uniformes em todas as plataformas e definindo uma aparência e comportamento uniformes entre as plataformas.



Aparência e comportamento plugáveis

- A classe ***UIManager.LookAndFeelInfo*** mantém as informações sobre uma aparência e um comportamento.

- O método ***static UIManager getInstalledLookAndFeels*** retorna um array de objetos ***UIManager.LookAndFeelInfo*** que descrevem as aparências e comportamentos disponíveis.

Aparência e comportamento plugáveis

- O método ***UIManager static setLookAndFeel*** altera a aparência e comportamento.

- O método ***SwingUtilities static updateComponentTreeUI*** altera a aparência e comportamento de cada componente associado ao seu argumento Component para a nova aparência e comportamento.

Exemplo - LookAndFeel

```
1 // Figura 22.9: LookAndFeelFrame.java
2 // Alterando a aparência e o comportamento.
3 import java.awt.GridLayout;
4 import java.awt.BorderLayout;
5 import java.awt.event.ItemListener;
6 import java.awt.event.ItemEvent;
7 import javax.swing.JFrame;
8 import javax.swing.UIManager;
9 import javax.swing.JRadioButton;
10 import javax.swing.ButtonGroup;
11 import javax.swing.JButton;
12 import javax.swing.JLabel;
13 import javax.swing.JComboBox;
14 import javax.swing.JPanel;
15 import javax.swing.SwingConstants;
16 import javax.swing.SwingUtilities;
17
```

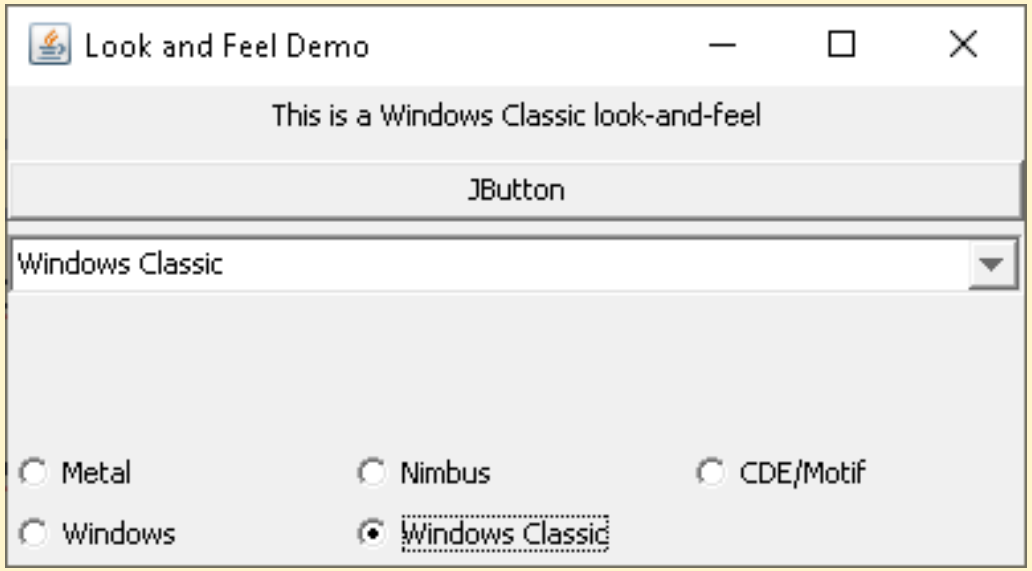
```
18 public class LookAndFeelFrame extends JFrame
19 {
20     private final UIManager.LookAndFeelInfo[] looks;
21     private final String[] lookNames;    // nomes da aparência e do comportamento
22     private final JRadioButton[] radio;  // para selecionar a aparência e o comport.
23     private final ButtonGroup group;     // grupo para botões de rádio
24     private final JButton button;        // exibe a aparência do botão
25     private final JLabel label;          // exibe a aparência do rótulo
26     private final JComboBox<String> comboBox; // exibe a aparência do comboBox
27
28     // configura a GUI
29     public LookAndFeelFrame()
30     {
31        super("Look and Feel Demo");
32
33        // obtém as informações sobre a aparência e comportamento instaladas
34        looks = UIManager.getInstalledLookAndFeels();
35        lookNames = new String[looks.length];
36
37        // obtém os nomes das aparências e comportamentos instalados
38        for (int i = 0; i < looks.length; i++)
39           lookNames[i] = looks[i].getName();
40
```

```
41      JPanel northPanel = new JPanel();
42      northPanel.setLayout(new GridLayout(3, 1, 0, 5));
43
44      label = new JLabel("This is a " + lookNames[0] + " look-and-feel",
45                  SwingConstants.CENTER);
46      northPanel.add(label);
47
48      button = new JButton("JButton");
49      northPanel.add(button);
50
51      comboBox = new JComboBox<String>(lookNames);
52      northPanel.add(comboBox);
53
54      // cria um array para botões de opção
55      radio = new JRadioButton[looks.length];
56
57      JPanel southPanel = new JPanel();
58
59      // usa um GridLayout com três botões em cada linha
60      int rows = (int) Math.ceil(radio.length / 3.0);
61      southPanel.setLayout(new GridLayout(rows, 3));
62
63      group = new ButtonGroup(); // grupo de botões para aparências e comportamentos
64      ItemHandler handler = new ItemHandler(); // rotina de tratamento ItemListener
```

```
65
66       for (int count = 0; count < radio.length; count++)
67       {
68          radio[count] = new JRadioButton(lookNames[count]);
69          radio[count].addItemListener(handler); // adiciona rotina de tratamento
70          group.add(radio[count]);              // adiciona botão de opções ao grupo
71          southPanel.add(radio[count]);         // adiciona botão de opções ao painel
72       }
73
74       add(northPanel, BorderLayout.NORTH); // adiciona o painel North
75       add(southPanel, BorderLayout.SOUTH); // adiciona o painel South
76
77       radio[0].setSelected(true); // configura a seleção padrão
78 } // fim do construtor LookAndFeelFrame
79
80 // utiliza UIManager para alterar a aparência e comportamento da GUI
81 private void changeTheLookAndFeel(int value)
82 {
83       try // muda a aparência e comportamento
84       {
85          // configura a aparência e comportamento para esse aplicativo
86          UIManager.setLookAndFeel(looks[value].getClassName());
87
```

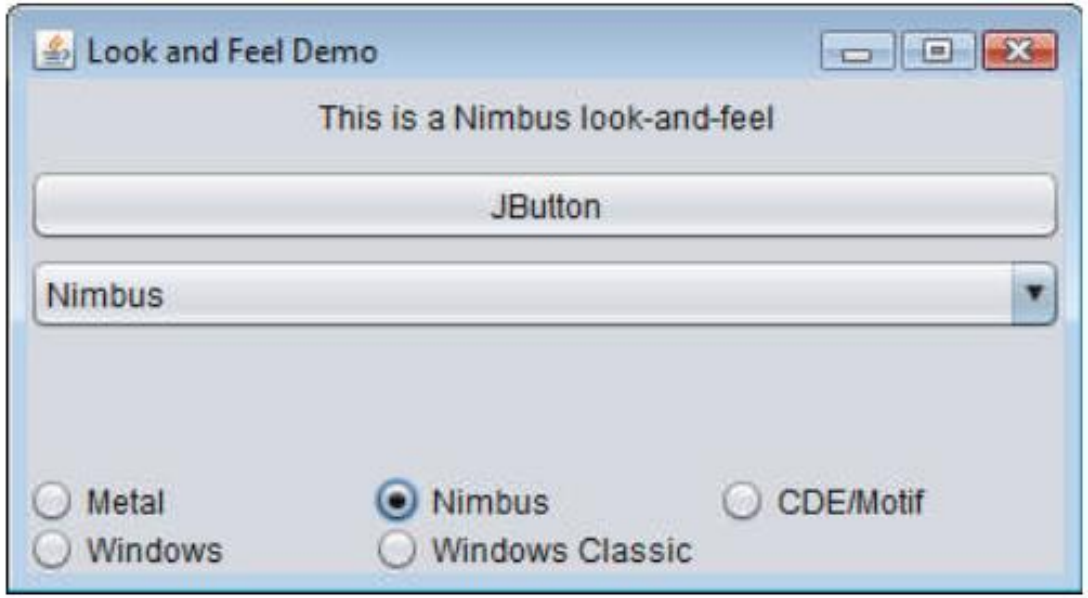
```
88          // atualiza os componentes nesse aplicativo
89          SwingUtilities.updateComponentTreeUI(this);
90       }
91       catch (Exception exception)
92       {
93          exception.printStackTrace();
94       }
95 }
96
97 // classe interna private para tratar eventos de botão de opção
98 private class ItemHandler implements ItemListener
99 {
100    // processa a seleção de aparência e comportamento feita pelo usuário
101    @Override
102    public void itemStateChanged(ItemEvent event)
103    {
104       for (int count = 0; count < radio.length; count++)
105       {
106          if (radio[count].isSelected())
107          {
108             label.setText(String.format(
109                               "This is a %s look-and-feel", lookNames[count]));
110             comboBox.setSelectedIndex(count); // configura o índice do comboBox
111             changeTheLookAndFeel(count);    // muda a aparência e comportamento
```

```
112            }
113         }
114      }
115   }  // fim da classe interna privada ItemHandler
116 }    // fim da classe LookAndFeelFrame
```

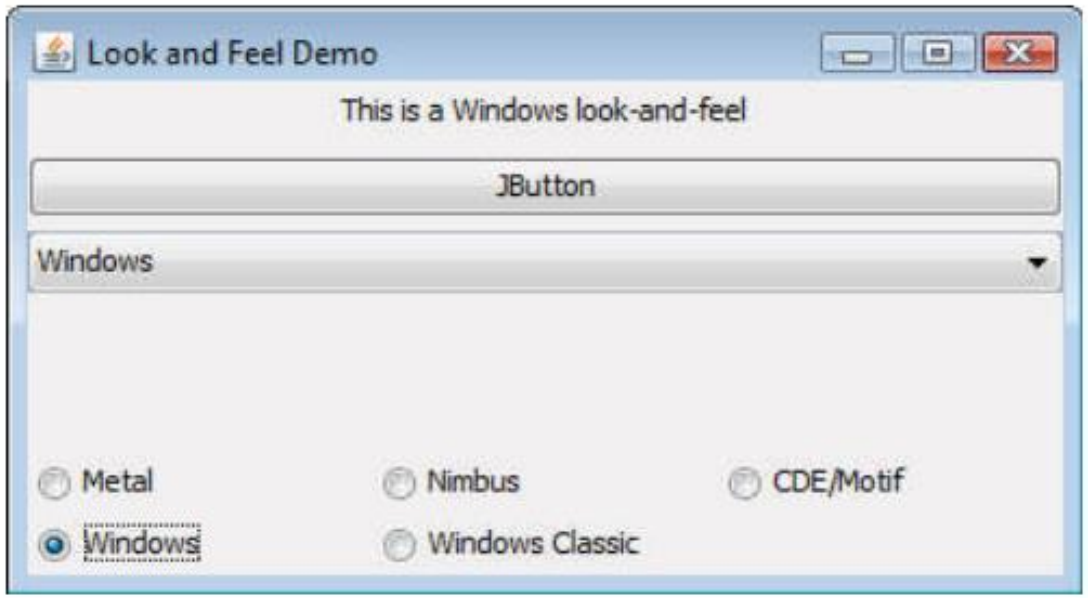


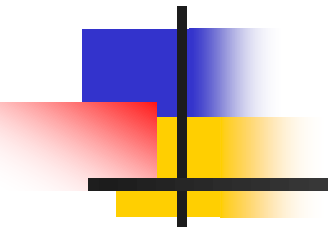

```
1 // Figura 22.10: LookAndFeelDemo.java
2 // Alterando a aparência e comportamento.
3 import javax.swing.JFrame;
4
5 public class LookAndFeelDemo
6 {
7    public static void main(String[] args)
8    {
9       LookAndFeelFrame lookAndFeelFrame = new LookAndFeelFrame();
10         lookAndFeelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11         lookAndFeelFrame.setSize(400, 220);
12         lookAndFeelFrame.setVisible(true);
13      }
14 } // fim da classe LookAndFeelDemo
```


Look and Feel Demo
This is a Metal look-and-feel
JButton
Metal
Metal
Nimbus
CDE/Motif
Windows
Windows Classic


Look and Feel Demo
This is a Nimbus look-and-feel
JButton
Nimbus
Metal
Nimbus
CDE/Motif
Windows
Windows Classic

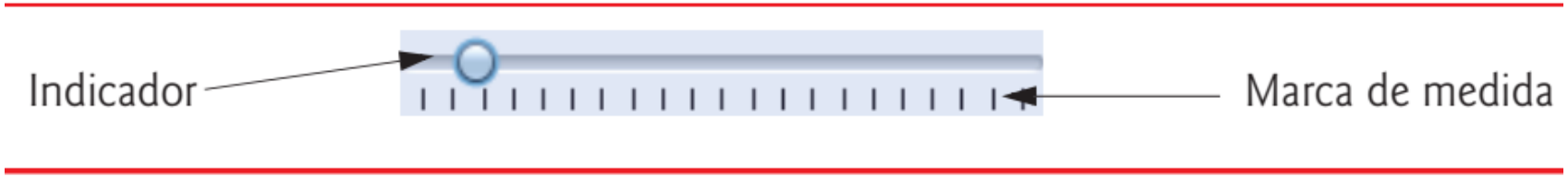

Look and Feel Demo
This is a CDE/Motif look-and-feel
JButton
CDE/Motif
Metal
Nimbus
CDE/Motif
Windows
Windows Classic


Look and Feel Demo
This is a Windows look-and-feel
JButton
Windows
Metal
Nimbus
CDE/Motif
Windows
Windows Classic

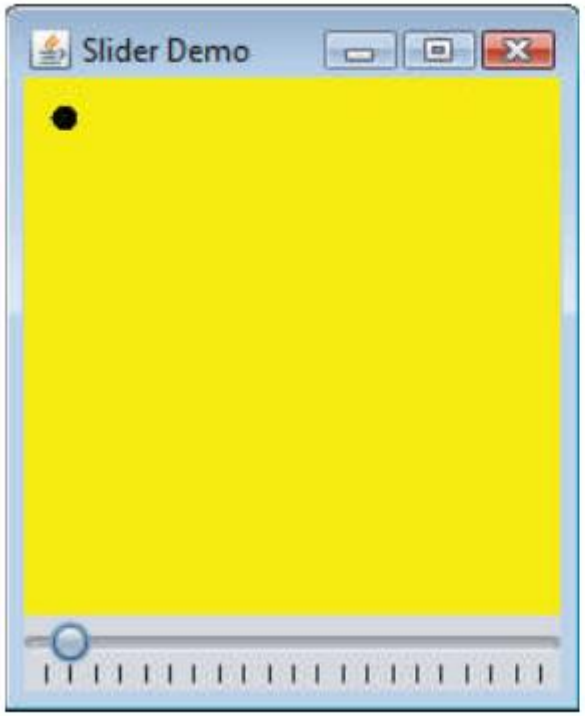


JSlider

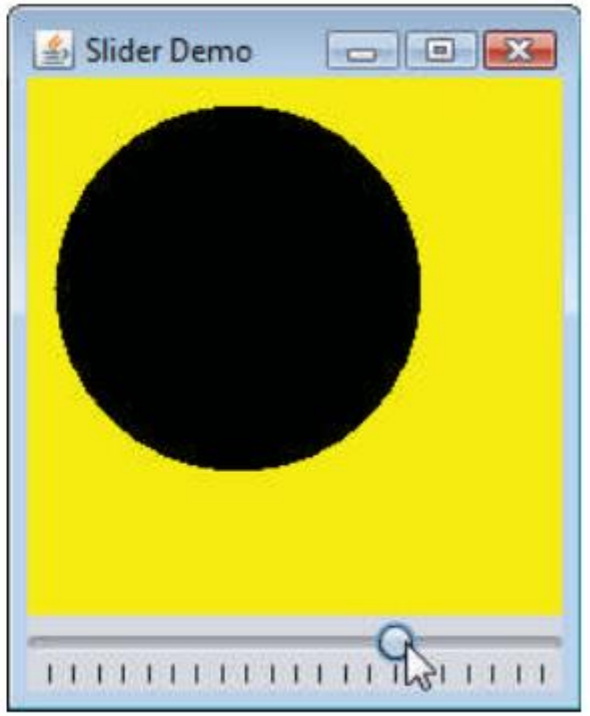
JSlider



a) GUI inicial com círculo padrão

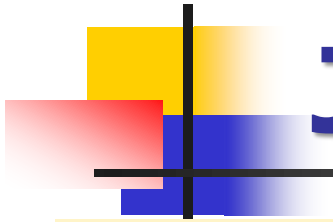


b) GUI depois que o usuário move o indicador de `JSlider` para a direita



JSlider

- **JSliders** permitem selecionar a partir de um intervalo de valores inteiros. Eles podem exibir marcas de medida significativas e secundárias, e rótulos para marcas de verificação.
- Os **JSliders** têm orientação horizontal ou vertical. Para um JSlider horizontal, o valor mínimo está na extrema esquerda e o valor máximo, na extrema direita. Para um **JSlider** vertical, o valor mínimo está na extremidade inferior e o valor máximo na extremidade superior.



JSlider

- A posição do indicador aponta o valor atual do **JSlider**. O método **getValue** da classe JSlider retorna a posição do indicador atual.
- O método **JSlider.setMajorTickSpacing ()** define o espaçamento para marcas de medida em um JSlider.
- O método **JSlider.setPaintTicks** com um argumento **true** indica que as marcas de medida devem ser exibidas.
- Os JSliders geram **ChangeEvents** quando o usuário interage com um JSlider. Um **ChangeListener** declara o método **stateChanged** que pode responder a **ChangeEvents**.

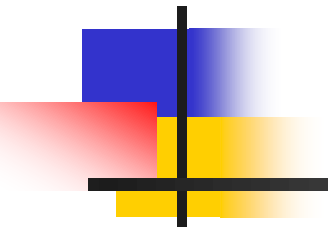
```java
// Figura 22.2: OvalPanel.java
// Uma classe JPanel personalizada.
import java.awt.Graphics;
import java.awt.Dimension;
import javax.swing.JPanel;

public class OvalPanel extends JPanel
{
   private int diameter = 10; // diâmetro padrão

   // desenha uma oval do diâmetro especificado
   @Override
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      g.fillOval(10, 10, diameter, diameter);
   }

```

```
19 // valida e configura o diâmetro e então repinta
20 public void setDiameter(int newDiameter)
21 {
22     // se diâmetro inválido, assume o padrão de 10
23     diameter = (newDiameter >= 0 ? newDiameter : 10);
24     repaint(); // repinta o painel
25 }
26
27 // utilizado pelo gerenciador de layout para determinar o tamanho preferido
28 public Dimension getPreferredSize()
29 {
30      return new Dimension(200, 200);
31 }
32
33 // utilizado pelo gerenciador de layout para determinar o tamanho mínimo
34 public Dimension getMinimumSize()
35 {
36     return getPreferredSize();
37 }
38 } // fim da classe OvalPanel
```

```
1 // Figura 22.3: SliderFrame.java
2 // Utilizando JSliders para dimensionar uma oval.
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import javax.swing.JFrame;
6 import javax.swing.JSlider;
7 import javax.swing.SwingConstants;
8 import javax.swing.event.ChangeListener;
9 import javax.swing.event.ChangeEvent;
10
11 public class SliderFrame extends JFrame
12 {
13    private final JSlider diameterJSlider; // slider para selecionar o diâmetro
14    private final OvalPanel myPanel; // painel para desenhar um círculo
15
16    // construtor sem argumento
17    public SliderFrame()
18    {
19       super("Slider Demo");
20
21       myPanel = new OvalPanel(); // cria o painel para desenhar um círculo
22       myPanel.setBackground(Color.YELLOW);
23
```

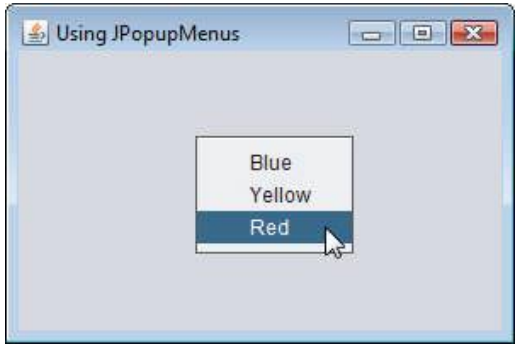
```
24      // configura o JSlider para controlar o valor do diâmetro
25      diameterJSlider =
26              new JSlider(SwingConstants.HORIZONTAL, 0, 200, 10);
27      diameterJSlider.setMajorTickSpacing(10); // cria uma marca de medida a cada 10
28      diameterJSlider.setPaintTicks(true); // pinta as marcas de medida no slider
29
30      // registra o ouvinte de evento do JSlider
31      diameterJSlider.addChangeListener(
32            new ChangeListener() // classe interna anônima
33              {
34                 // trata da alteração de valor do controle deslizante
35                 @Override
36                 public void stateChanged(ChangeEvent e)
37                 {
38                    myPanel.setDiameter(diameterJSlider.getValue());
39                 }
40              }
41         );
42
43      add(diameterJSlider, BorderLayout.SOUTH);
44      add(myPanel, BorderLayout.CENTER);
45   }
46 } // fim da classe SliderFrame
```

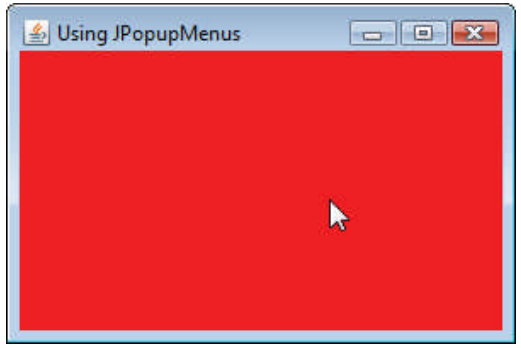
```
1 // Figura 22.4: SliderDemo.java
2 // Testando SliderFrame.
3 import javax.swing.JFrame;
4
5 public class SliderDemo
6 {
7    public static void main(String[] args)
8    {
9       SliderFrame sliderFrame = new SliderFrame();
10      sliderFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11      sliderFrame.setSize(220, 270);
12      sliderFrame.setVisible(true);
13   }
14 } // fim da classe SliderDemo
```



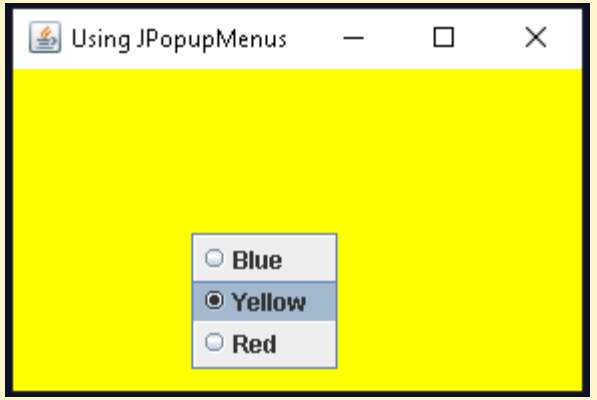
JPopupMenu

JPopupMenu





- Menus pop-up sensíveis ao contexto são criados com a classe **JPopupMenu**. O evento de acionamento do pop-up normalmente ocorre quando o usuário pressiona e libera o botão direito do mouse. O método **MouseEvent.isPopupTrigger** retorna *true* se o evento de acionamento do pop-up ocorreu.
- O método **JPopupMenu.show** exibe um *JPopupMenu*. O primeiro argumento especifica o componente de origem, que ajuda a determinar onde o *JPopupMenu* aparecerá. Os dois últimos argumentos são as coordenadas no canto superior esquerdo do componente de origem, em que o *JPopupMenu* aparece.



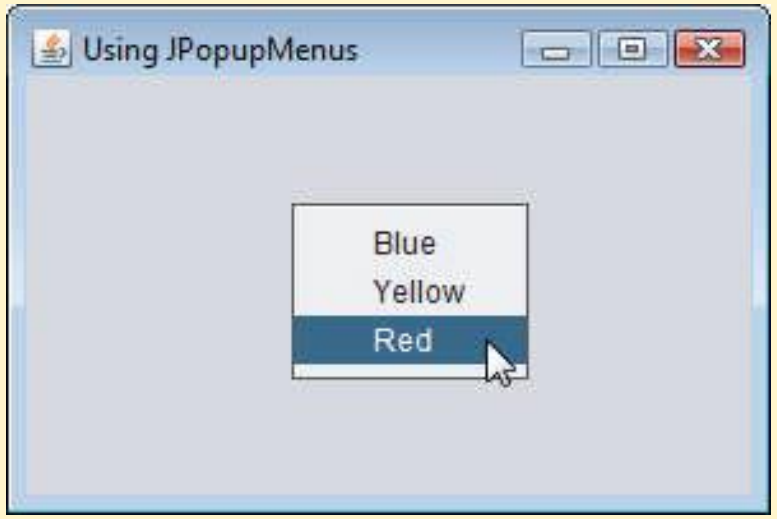

```
1  // Figura 22.7: PopupFrame.java
2  // Demonstrando JPopupMenus.
3  import java.awt.Color;
4  import java.awt.event.MouseAdapter;
5  import java.awt.event.MouseEvent;
6  import java.awt.event.ActionListener;
7  import java.awt.event.ActionEvent;
8  import javax.swing.JFrame;
9  import javax.swing.JRadioButtonMenuItem;
10 import javax.swing.JPopupMenu;
11 import javax.swing.ButtonGroup;
12
13 public class PopupFrame extends JFrame
14 {
15    private final JRadioButtonMenuItem[] items; // contém itens para cores
16    private final Color[] colorValues =
17          { Color.BLUE, Color.YELLOW, Color.RED }; // cores a serem utilizadas
18    private final JPopupMenu popupMenu; // permite que o usuário selecione a cor
19
20    // construtor sem argumento configure a GUI
21    public PopupFrame()
22    {
```

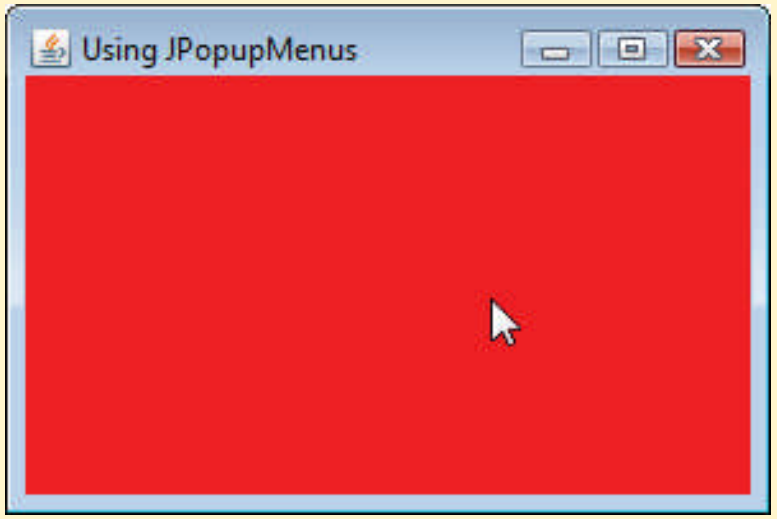
```
23      super("Using JPopupMenus");
24
25      ItemHandler handler = new ItemHandler(); // classe de trat. para itens de menu
26      String[] colors = { "Blue", "Yellow", "Red" };
27
28      ButtonGroup colorGroup = new ButtonGroup();  // gerencia itens de cor
29      popupMenu = new JPopupMenu();                // cria menu pop-up
30      items = new JRadioButtonMenuItem[colors.length];
31
32      // cria item de menu, adiciona-o ao menu pop-up, permite tratamento de eventos
33      for (int count = 0; count < items.length; count++)
34      {
35         items[count] = new JRadioButtonMenuItem(colors[count]);
36         popupMenu.add(items[count]); // adiciona o item ao menu pop-up
37         colorGroup.add(items[count]); // adiciona o item ao grupo de botões
38         items[count].addActionListener(handler); // adiciona handler
39      }
40
41      setBackground(Color.WHITE);
42
43      // declara um MouseListener para a janela a fim de exibir o menu pop-up
44      addMouseListener(
45              new MouseAdapter() // classe interna anônima
46              {
47              // trata eventos de pressionamento do mouse
```

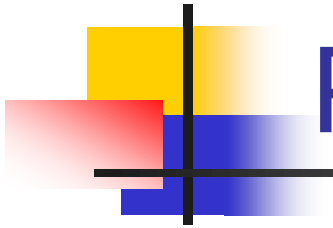
```
48              @Override
49                 public void mousePressed(MouseEvent event)
50                 {
51                    checkForTriggerEvent(event);
52                 }
53
54                 // trata eventos de liberação de botão do mouse
55                 @Override
56                 public void mouseReleased(MouseEvent event)
57                 {
58                    checkForTriggerEvent(event);
59                 }
60
61                 // determina se o evento deve acionar o menu pop-up
62                 private void checkForTriggerEvent(MouseEvent event)
63                 {
64                    if (event.isPopupTrigger())
65                       popupMenu.show(
66                             event.getComponent(), event.getX(), event.getY());
67                 }
68           }
69        );
70    } // fim do construtor PopupFrame
```

```
71
72    // classe interna privada para tratar eventos de item de menu
73    private class ItemHandler implements ActionListener
74    {
75       // processa seleções de itens de menu
76       @Override
77       public void actionPerformed(ActionEvent event)
78       {
79          // determina qual item de menu foi selecionado
80          for (int i = 0; i < items.length; i++)
81          {
82             if (event.getSource() == items[i])
83             {
84                getContentPane().setBackground(colorValues[i]);
85                return;
86             }
87          }
88       }
89    } // fim da classe interna privada ItemHandler
90 } // fim da classe PopupFrame
```

```java
// Figura 22.8: PopupTest.java
// Testando PopupFrame.
import javax.swing.JFrame;

public class PopupTest
{
   public static void main(String[] args)
   {
      PopupFrame popupFrame = new PopupFrame();
      popupFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      popupFrame.setSize(300, 200);
      popupFrame.setVisible(true);
   }
} // fim da classe PopupTest
```







Referencias

- Java How to program 3 a 10 ed.

Deitel e Deitel