Interface Gráfica

Using Inheritance to Customize Panels

- Applets: Can generate drawings by overriding `paint`
- Doesn't work for Swing components (e.g. `JPanel`)
- Must override `paintComponent` instead
- Important: Call `super.paintComponent` to paint the background
-
```
public class MyPanel
{
    public void paintComponent(Graphics g)
    {
          super.paintComponent(g);
          Graphics2D g2 = (Graphics2D)g;
          . . .
    }
}
```

- Default panel size is 0 x 0 pixels. Change in constructor:

```
public MyPanel()
{
    setPreferredSize(new Dimension(width, height));
    . . .
}
```

Turn Rectangle Applet into Application

- **Make class** `RectanglePanel`
- `paintComponent` **method paints the rectangle at current location**
- Panel should store the data that it needs to paint itself

```
public class RectanglePanel
{
   . . .
   private Rectangle box;
}
```

- **Mouse listener changes location of rectangle**

File RectangleTest.java

```
1 import javax.swing.JButton;
2 import javax.swing.JFrame;
3 import javax.swing.JLabel;
4 import javax.swing.JPanel;
5 import javax.swing.JTextField;
6
7 /**
8    This program displays a frame containing a RectanglePanel.
9 */
10 public class RectangleTest
11 {
12    public static void main(String[] args)
13    {
14       RectanglePanel rectPanel = new RectanglePanel();
15
16       JFrame appFrame = new JFrame();
17       appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
18       appFrame.setContentPane(rectPanel);
19       appFrame.pack();
20       appFrame.show();
21    }
22 }
```

File RectanglePanel.java

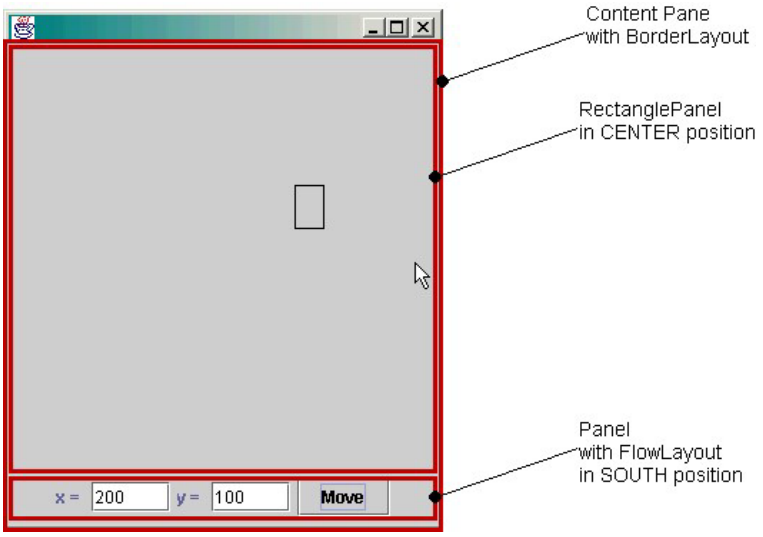
```
1 import java.awt.event.MouseEvent;
2 import java.awt.event.MouseListener;
3 import java.awt.Dimension;
4 import java.awt.Graphics;
5 import java.awt.Graphics2D;
6 import java.awt.Rectangle;
7 import javax.swing.JPanel;
8
9 /**
10    A rectangle panel displays a rectangle that a user can
11    move by clicking the mouse.
12 */
13 public class RectanglePanel extends JPanel
14 {
15    /**
16       Constructs a rectangle panel with the rectangle at a
17       default location.
```

```
18    */
19    public RectanglePanel()
20    {
21       setPreferredSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
22
23       // the rectangle that the paint method draws
24       box = new Rectangle(BOX_X, BOX_Y,
25          BOX_WIDTH, BOX_HEIGHT);
26
27       // add mouse press listener
28
29       class MousePressListener implements MouseListener
30       {
31          public void mousePressed(MouseEvent event)
32          {
33             int x = event.getX();
34             int y = event.getY();
35             box.setLocation(x, y);
36             repaint();
37          }
```

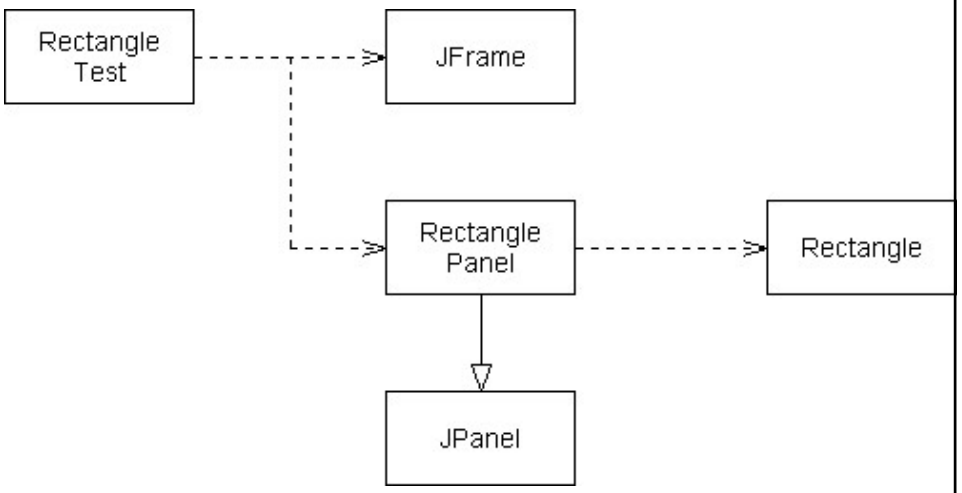
```
38
39        //  do-nothing methods
40        public void mouseReleased(MouseEvent event) {}
41        public void mouseClicked(MouseEvent event) {}
42        public void mouseEntered(MouseEvent event) {}
43        public void mouseExited(MouseEvent event) {}
44     }
45
46     MouseListener listener = new MousePressListener();
47     addMouseListener(listener);
48  }
49
50  public void paintComponent(Graphics g)
51  {
52     super.paintComponent(g);
53     Graphics2D g2 = (Graphics2D)g;
54     g2.draw(box);
55  }
56
57  private Rectangle box;
```

```
58     private static final int BOX_X = 100;
59     private static final int BOX_Y = 100;
60     private static final int BOX_WIDTH = 20;
61     private static final int BOX_HEIGHT = 30;
62
63     private static final int PANEL_WIDTH = 300;
64     private static final int PANEL_HEIGHT = 300;
65  }
```

The Layout of the RectanglePanel



Classes of the Rectangle Application

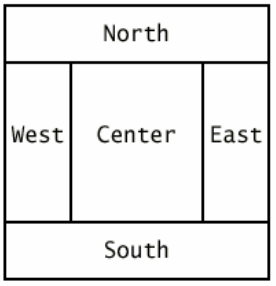


Layout Management

- **Container arranges its components**
- **By default, `JPanel` places components from left to right, then starts a new row if needed**
- **Panel layout carried out by `FlowLayout` layout manager**
- **Can set other layout managers**
  `panel.setLayout(new BorderLayout());`

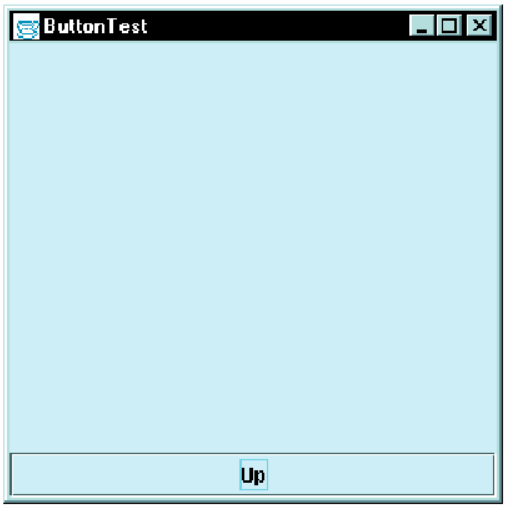
Border Layout

- Border layout places components into positions (center, north, west, south, east)
  `panel.add(textField, BorderLayout.SOUTH);`



- Content pane of frame has border layout by default
  `frame.getContentPane().add(textField, BorderLayout.SOUTH);`
- Border layout grows components to fit area
- To avoid growth, place component into panel (with flow layout), then add panel to content pane

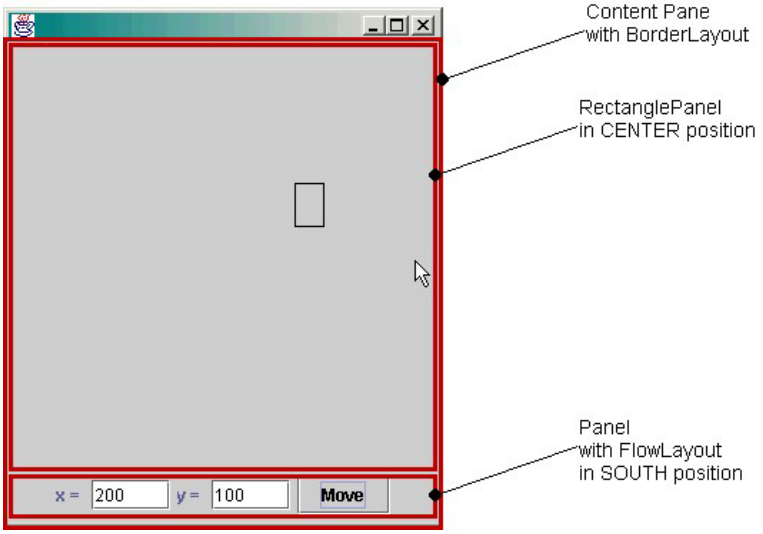
Components Expand to Fill BorderLayout Area



Grid Layout

- **Lays out components in rows and columns**
- **All components have the same size**
- **Add components left to right (top row first, the second row, etc.)**
- ```
  panel.setLayout(new GridLayout(4, 3));
  panel.add(button7);
  panel.add(button8);
  panel.add(button9);
  panel.add(button6);
  ```
  ```

Combining Layout Managers



Using Inheritance to Customize Frames

- Use inheritance to define complex frames
- ```
  public class RectangleFrame extends JFrame
  {
     public RectangleFrame()
     {
        add components to content pane
     }
  }
  ```
- Avoid lengthy `main` method
- Add methods to set up panels etc.
- Turn GUI components into instance variables

Using Inheritance to Customize Frames

- **Use separate test class**

```
public class RectangleTest
{
   public static void main(String[] args)
   {
      construct and display frame
   }
}
```
```

- **Frame handlers (e.g. buttons, menus) call methods to communicate changes to panel**

```
class RectanglePanel extends Panel
{
   . . .
   public void setLocation(int x, int y)
   {
      box.setLocation(x, y);
      repaint();
   }
}
```

File RectangleTest.java

```
1 import javax.swing.JFrame;
2
3 /**
4    This program tests the RectangleFrame.
5 */
6 public class RectangleTest
7 {
8    public static void main(String[] args)
9    {
10      JFrame appFrame = new RectangleFrame();
11      appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12      appFrame.show();
13   }
14 }
```

File RectangleFrame.java

```
1 import java.awt.BorderLayout;
2 import java.awt.event.ActionEvent;
3 import java.awt.event.ActionListener;
4 import javax.swing.JButton;
5 import javax.swing.JFrame;
6 import javax.swing.JLabel;
7 import javax.swing.JPanel;
8 import javax.swing.JTextField;
9
10 /**
11    This frame contains a panel that displays a rectangle
12    and a panel of text fields to specify the rectangle position.
13 */
14 public class RectangleFrame extends JFrame
15 {
16    /**
17       Constructs the frame.
```

```
18    */
19    public RectangleFrame()
20    {
21       // the panel that draws the rectangle
22       rectPanel = new RectanglePanel();
23
24       // add panel to content Pane
25       getContentPane().add(rectPanel, BorderLayout.CENTER);
26
27       createControlPanel();
28
29       pack();
30    }
31
32    /**
33       Creates the control panel with the text fields
34       at the bottom of the frame.
35    */
36    private void createControlPanel()
37    {
```

```
38      // the text fields for entering the x- and y-coordinates
39      final JTextField xField = new JTextField(5);
40      final JTextField yField = new JTextField(5);;
41
42      // the button to move the rectangle
43      JButton moveButton = new JButton("Move");
44
45      class MoveButtonListener implements ActionListener
46      {
47        public void actionPerformed(ActionEvent event)
48        {
49          int x = Integer.parseInt(xField.getText());
50          int y = Integer.parseInt(yField.getText());
51          rectPanel.setLocation(x, y);
52        }
53      };
54
55      ActionListener listener = new MoveButtonListener();
56      moveButton.addActionListener(listener);
57
```

```
58      // the labels for labeling the text fields
59      JLabel xLabel = new JLabel("x = ");
60      JLabel yLabel = new JLabel("y = ");
61
62      // the panel for holding the user interface components
63      JPanel controlPanel = new JPanel();
64
65      controlPanel.add(xLabel);
66      controlPanel.add(xField);
67      controlPanel.add(yLabel);
68      controlPanel.add(yField);
69      controlPanel.add(moveButton);
70
71      getContentPane().add(controlPanel, BorderLayout.SOUTH);
72    }
73
74    private RectanglePanel rectPanel;
75 }
76
77
```

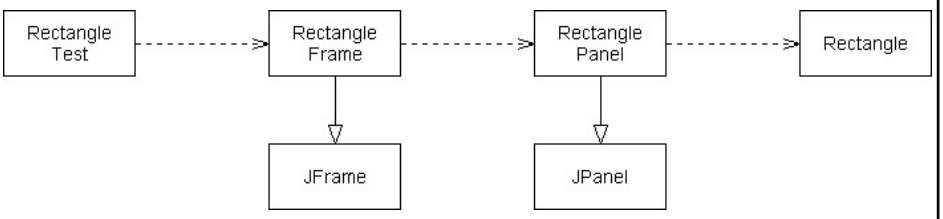
File RectanglePanel.java

```
1 import java.awt.Dimension;
2 import java.awt.Graphics;
3 import java.awt.Graphics2D;
4 import java.awt.Rectangle;
5 import javax.swing.JPanel;
6
7 /**
8    This panel displays a rectangle.
9 */
10 public class RectanglePanel extends JPanel
11 {
12    /**
13       Constructs a rectangle panel with the rectangle at a
14       default location.
15    */
16    public RectanglePanel()
17    {
```

```
18       setPreferredSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
19       // the rectangle that the paint method draws
20       box = new Rectangle(BOX_X, BOX_Y,
21          BOX_WIDTH, BOX_HEIGHT);
22    }
23
24    /**
25       Sets the location of the rectangle and repaints the panel.
26       @param x the x-coordinate of the top left corner of the rectangle
27       @param y the y-coordinate of the top left corner of the rectangle
28    */
29    public void setLocation(int x, int y)
30    {
31       box.setLocation(x, y);
32       repaint();
33    }
34
35    public void paintComponent(Graphics g)
36    {
37       super.paintComponent(g);
```

```
38      Graphics2D g2 = (Graphics2D)g;
39      g2.draw(box);
40    }
41
42    private Rectangle box;
43    private static final int BOX_X = 100;
44    private static final int BOX_Y = 100;
45    private static final int BOX_WIDTH = 20;
46    private static final int BOX_HEIGHT = 30;
47
48    private static final int PANEL_WIDTH = 300;
49    private static final int PANEL_HEIGHT = 300;
50 }
```

Classes in the Rectangle Frame Application



Radio Buttons

- **Radio buttons are mutually exclusive**
- **Button group turns one button off when the next one is turned on**

```
sButton = new JRadioButton("Small");
mButton = new JRadioButton("Medium");
lButton = new JRadioButton("Large");
ButtonGroup group = new ButtonGroup();
group.add(sbutton);
group.add(mbutton);
group.add(lbutton);
```

- **Button group doesn't place buttons into panel--need to add them:**

```
panel.add(sButton);
```

- **In action listener, test if selected**

```
if (sButton.isSelected()) . . .
```

A Combo Box, Check Boxes, and Radio Buttons

Check Boxes

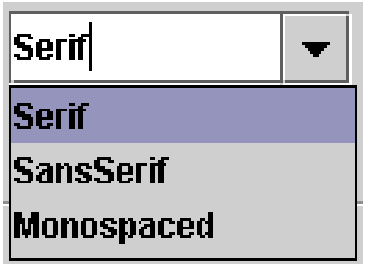
- Similar to radio button, not mutually exclusive

```
JCheckBox it = new JCheckBox("Italic");
```

- Don't place into button group

Combo Boxes

- Use less space than radio buttons
- Users can type other values
- “Combo” between list selection and text field



- JComboBox faceName = new JComboBox();
  faceName.addItem("Serif");
  faceName.addItem("SansSerif");
  . . .
- Get user selection
  sel= (String)faceName.getSelectedItem();

File ChoiceTest.java

```
1 import javax.swing.JFrame;
2
3 /**
4    This program tests the ChoiceFrame.
5 */
6 public class ChoiceTest
7 {
8    public static void main(String[] args)
9    {
10       JFrame frame = new ChoiceFrame();
11       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12       frame.show();
13    }
14 }
15
```

File ChoiceFrame.java

```
1 import java.awt.BorderLayout;
2 import java.awt.Container;
3 import java.awt.Font;
4 import java.awt.GridLayout;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import java.awt.event.WindowAdapter;
8 import java.awt.event.WindowEvent;
9 import javax.swing.ButtonGroup;
10 import javax.swing.JButton;
11 import javax.swing.JCheckBox;
12 import javax.swing.JComboBox;
13 import javax.swing.JFrame;
14 import javax.swing.JLabel;
15 import javax.swing.JPanel;
16 import javax.swing.JRadioButton;
17 import javax.swing.border.EtchedBorder;
```

```
18 import javax.swing.border.TitledBorder;
19
20 /**
21    This frame contains a text field and a control panel
22    to change the font of the text.
23 */
24 public class ChoiceFrame extends JFrame
25 {
26    /**
27       Constructs the frame.
28    */
29    public ChoiceFrame()
30    {
31       // construct text sample
32       sampleField = new JLabel("Big Java");
33       getContentPane().add(sampleField, BorderLayout.CENTER);
34
35       // this listener is shared among all components
36       class ChoiceListener implements ActionListener
37       {
```

```
38          public void actionPerformed(ActionEvent event)
39          {
40             setSampleFont();
41          }
42       }
43
44       listener = new ChoiceListener();
45
46       createControlPanel();
47       setSampleFont();
48       pack();
49    }
50
51    /**
52       Creates the control panel to change the font.
53    */
54    public void createControlPanel()
55    {
56       JPanel facenamePanel = createComboBox();
57       JPanel sizeGroupPanel = createCheckBoxes();
```

```
58      JPanel styleGroupPanel = createRadioButtons();
59
60      // line up component panels
61
62      JPanel controlPanel = new JPanel();
63      controlPanel.setLayout(new GridLayout(3, 1));
64      controlPanel.add(facenamePanel);
65      controlPanel.add(sizeGroupPanel);
66      controlPanel.add(styleGroupPanel);
67
68      // add panels to content pane
69
70      getContentPane().add(controlPanel, BorderLayout.SOUTH);
71   }
72
73   /**
74      Creates the combo box with the font style choices.
75      @return the panel containing the combo box
76   */
77   public JPanel createComboBox()
```

```
78   {
79      facenameCombo = new JComboBox();
80      facenameCombo.addItem("Serif");
81      facenameCombo.addItem("SansSerif");
82      facenameCombo.addItem("Monospaced");
83      facenameCombo.setEditable(true);
84      facenameCombo.addActionListener(listener);
85
86      JPanel panel = new JPanel();
87      panel.add(facenameCombo);
88      return panel;
89   }
90
91   /**
92      Creates the check boxes for selecting bold and italic style
93      @return the panel containing the check boxes
94   */
95   public JPanel createCheckBoxes()
96   {
97      italicCheckBox = new JCheckBox("Italic");
```

```
98      italicCheckBox.addActionListener(listener);
99
100     boldCheckBox = new JCheckBox("Bold");
101     boldCheckBox.addActionListener(listener);
102
103     JPanel panel = new JPanel();
104     panel.add(italicCheckBox);
105     panel.add(boldCheckBox);
106     panel.setBorder
107        (new TitledBorder(new EtchedBorder(), "Style"));
108
109     return panel;
110  }
111
112  /**
113     Creates the radio buttons to select the font size
114     @return the panel containing the radio buttons
115  */
116  public JPanel createRadioButtons()
117  {
```

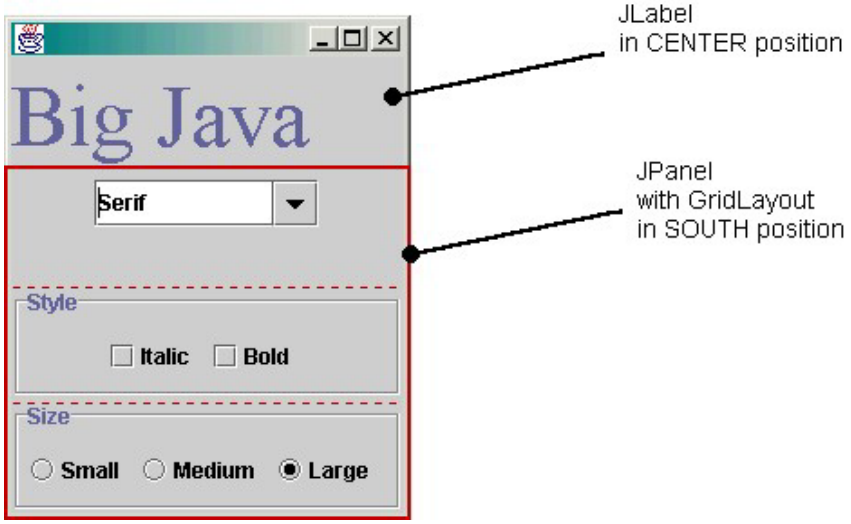
```
118     smallButton = new JRadioButton("Small");
119     smallButton.addActionListener(listener);
120
121     mediumButton = new JRadioButton("Medium");
122     mediumButton.addActionListener(listener);
123
124     largeButton = new JRadioButton("Large");
125     largeButton.addActionListener(listener);
126     largeButton.setSelected(true);
127
128     // add radio buttons to button group
129
130     ButtonGroup group = new ButtonGroup();
131     group.add(smallButton);
132     group.add(mediumButton);
133     group.add(largeButton);
134
135     JPanel panel = new JPanel();
136     panel.add(smallButton);
137     panel.add(mediumButton);
```

```
138     panel.add(largeButton);
139     panel.setBorder
140        (new TitledBorder(new EtchedBorder(), "Size"));
141
142     return panel;
143  }
144
145  /**
146     Gets user choice for font name, style, and size
147     and sets the font of the text sample.
148  */
149  public void setSampleFont()
150  {  // get font name
151
152     String facename
153        = (String)facenameCombo.getSelectedItem();
154
155     // get font style
156
157     int style = 0;
```

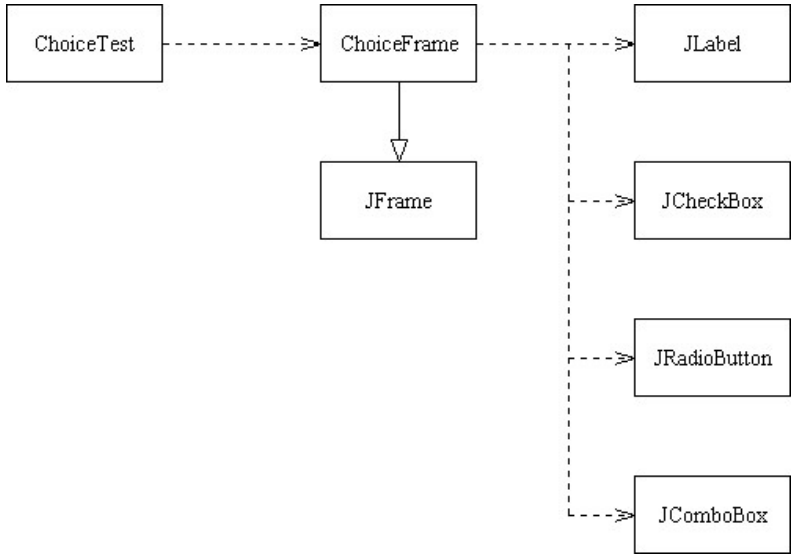
```
158     if (italicCheckBox.isSelected())
159        style = style + Font.ITALIC;
160     if (boldCheckBox.isSelected())
161        style = style + Font.BOLD;
162
163     // get font size
164
165     int size = 0;
166
167     final int SMALL_SIZE = 24;
168     final int MEDIUM_SIZE = 36;
169     final int LARGE_SIZE = 48;
170
171     if (smallButton.isSelected())
172        size = SMALL_SIZE;
173     else if (mediumButton.isSelected())
174        size = MEDIUM_SIZE;
175     else if (largeButton.isSelected())
176        size = LARGE_SIZE;
177
```

```
178      // set font of text field
179
180      sampleField.setFont(new Font(facename, style, size));
181      sampleField.repaint();
182   }
183
184   private JLabel sampleField;
185   private JCheckBox italicCheckBox;
186   private JCheckBox boldCheckBox;
187   private JRadioButton smallButton;
188   private JRadioButton mediumButton;
189   private JRadioButton largeButton;
190   private JComboBox facenameCombo;
191   private ActionListener listener;
192 }
```

The Components of the ChoiceFrame



Classes of the ChoiceTest Program



Menus

- **Add menu bar to frame**

```
JMenuBar bar = new JMenuBar();
frame.setJMenuBar(bar);
```

- **Add menus to the menu bar**

```
JMenu fileMenu
   = new JMenu("File");
bar.add(fileMenu);
```

Menu Items

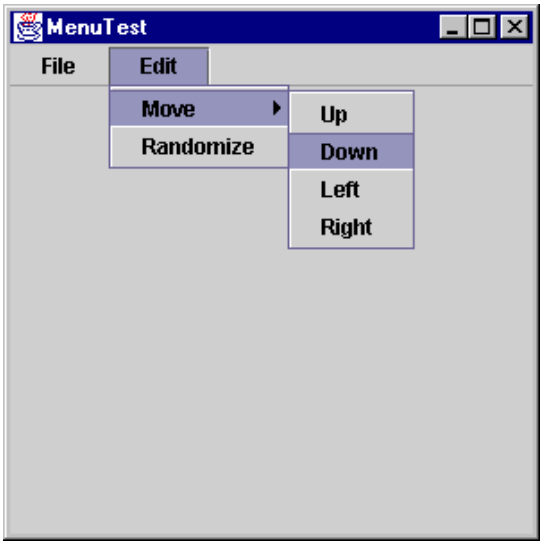
- **Add menu items to the menu**

```
JMenuItem fileNew
   = new JMenuItem("New");
```

- **Add action listener to the menu item**

```
ActionListener l = ...;
fileNew.addActionListener(l);
```

Pull-Down Menus



File MenuTest.java

```
1 import javax.swing.JFrame;
2
3 /**
4    This program tests the MenuFrame.
5 */
6 public class MenuTest
7 {
8    public static void main(String[] args)
9    {
10      JFrame frame = new MenuFrame();
11      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12      frame.show();
13   }
14 }
15
```

File MenuFrame.java

```
1 import java.awt.BorderLayout;
2 import java.awt.event.ActionEvent;
3 import java.awt.event.ActionListener;
4 import java.util.Random;
5 import javax.swing.JFrame;
6 import javax.swing.JMenu;
7 import javax.swing.JMenuBar;
8 import javax.swing.JMenuItem;
9
10 /**
11    This frame has a menu with commands to set the position of
12    a rectangle.
13 */
14 class MenuFrame extends JFrame
15 {
16    /**
17       Constructs the frame.
```

```
18   */
19   public MenuFrame()
20   {
21      generator = new Random();
22
23      // add drawing panel to content pane
24
25      panel = new RectanglePanel();
26      getContentPane().add(panel, BorderLayout.CENTER);
27      pack();
28
29      // construct menu
30
31      JMenuBar menuBar = new JMenuBar();
32      setJMenuBar(menuBar);
33
34      menuBar.add(createFileMenu());
35      menuBar.add(createEditMenu());
36   }
37
```

```
38   /**
39      Creates the File menu.
40      @return the menu
41   */
42   public JMenu createFileMenu()
43   {
44      JMenu menu = new JMenu("File");
45      menu.add(createFileNewItem());
46      menu.add(createFileExitItem());
47      return menu;
48   }
49
50   /**
51      Creates the Edit menu.
52      @return the menu
53   */
54   public JMenu createEditMenu()
55   {
56      JMenu menu = new JMenu("Edit");
57      menu.add(createMoveMenu());
```

```
58      menu.add(createEditRandomizeItem());
59      return menu;
60    }
61
62    /**
63      Creates the Move submenu.
64      @return the menu
65    */
66    public JMenu createMoveMenu()
67    {
68      JMenu menu = new JMenu("Move");
69      menu.add(createMoveItem("Up", 0, -1));
70      menu.add(createMoveItem("Down", 0, 1));
71      menu.add(createMoveItem("Left", -1, 0));
72      menu.add(createMoveItem("Right", 1, 0));
73      return menu;
74    }
75
76    /**
77      Creates the File->New menu item and sets its action listener.
```

```
78      @return the menu item
79    */
80    public JMenuItem createFileNewItem()
81    {
82      JMenuItem item = new JMenuItem("New");
83      class MenuItemListener implements ActionListener
84      {
85        public void actionPerformed(ActionEvent event)
86        {
87          panel.reset();
88        }
89      }
90      ActionListener listener = new MenuItemListener();
91      item.addActionListener(listener);
92      return item;
93    }
94
95    /**
96      Creates the File->Exit menu item and sets its action listener.
97      @return the menu item
```

```
98    */
99    public JMenuItem createFileExitItem()
100   {
101      JMenuItem item = new JMenuItem("Exit");
102      class MenuItemListener implements ActionListener
103      {
104         public void actionPerformed(ActionEvent event)
105         {
106            System.exit(0);
107         }
108      }
109      ActionListener listener = new MenuItemListener();
110      item.addActionListener(listener);
111      return item;
112   }
113
114   /**
115      Creates a menu item to move the rectangle and sets its
116      action listener.
117      @param label the menu label
```

```
118      @param dx the amount by which to move the rectangle in x-
      direction
119      @param dy the amount by which to move the rectangle in y-
      direction
120      @return the menu item
121   */
122   public JMenuItem createMoveItem(String label,
123      final int dx, final int dy)
124   {
125      JMenuItem item = new JMenuItem(label);
126      class MenuItemListener implements ActionListener
127      {
128         public void actionPerformed(ActionEvent event)
129         {
130            panel.moveRectangle(dx, dy);
131         }
132      }
133      ActionListener listener = new MenuItemListener();
134      item.addActionListener(listener);
135      return item;
136   }
137
```

```
138    /**
139       Creates the Edit->Randomize menu item and sets its action
       listener.
140       @return the menu item
141    */
142    public JMenuItem createEditRandomizeItem()
143    {
144       JMenuItem item = new JMenuItem("Randomize");
145       class MenuItemListener implements ActionListener
146       {
147          public void actionPerformed(ActionEvent event)
148          {
149             int width = panel.getWidth();
150             int height = panel.getHeight();
151             int dx = -1 + generator.nextInt(2);
152             int dy = -1 + generator.nextInt(2);
153             panel.moveRectangle(dx, dy);
154          }
155       }
156       ActionListener listener = new MenuItemListener();
157       item.addActionListener(listener);
```

```
158       return item;
159    }
160
161    private RectanglePanel panel;
162    private Random generator;
163 }
164
165
166
167
```

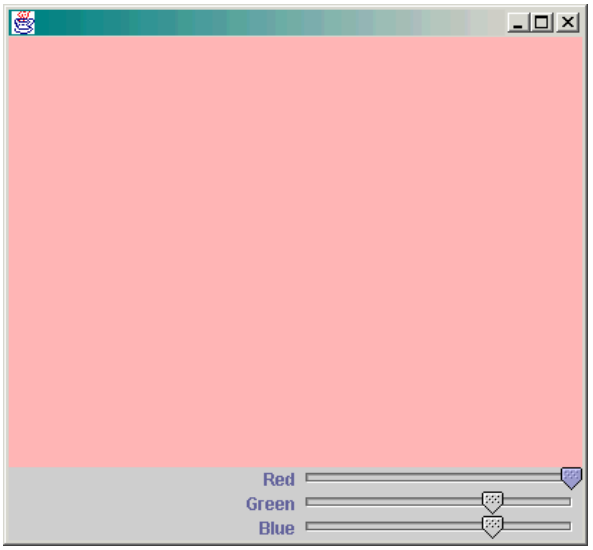
File RectanglePanel.java

```
1 import java.awt.Dimension;
2 import java.awt.Graphics;
3 import java.awt.Graphics2D;
4 import java.awt.Rectangle;
5 import javax.swing.JPanel;
6
7 /**
8    A panel that shows a rectangle.
9 */
10 class RectanglePanel extends JPanel
11 {
12    /**
13       Constructs a panel with the rectangle in the top left
14       corner.
15    */
16    public RectanglePanel()
17    {
```

```
18       setPreferredSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
19       // the rectangle that the paint method draws
20       box = new Rectangle(0, 0, BOX_WIDTH, BOX_HEIGHT);
21    }
22
23    public void paintComponent(Graphics g)
24    {
25       super.paintComponent(g);
26       Graphics2D g2 = (Graphics2D)g;
27       g2.draw(box);
28    }
29
30    /**
31       Resets the rectangle to the top left corner.
32    */
33    public void reset()
34    {
35       box.setLocation(0, 0);
36       repaint();
37    }
```

```
38
39  /**
40     Moves the rectangle and repaints it. The rectangle
41     is moved by multiples of its full width or height.
42     @param dx the number of width units
43     @param dy the number of height units
44  */
45  public void moveRectangle(int dx, int dy)
46  {
47     box.translate(dx * BOX_WIDTH, dy * BOX_HEIGHT);
48     repaint();
49  }
50
51  private Rectangle box;
52  private static final int BOX_WIDTH = 20;
53  private static final int BOX_HEIGHT = 30;
54  private static final int PANEL_WIDTH = 300;
55  private static final int PANEL_HEIGHT = 300;
56 }
57
```

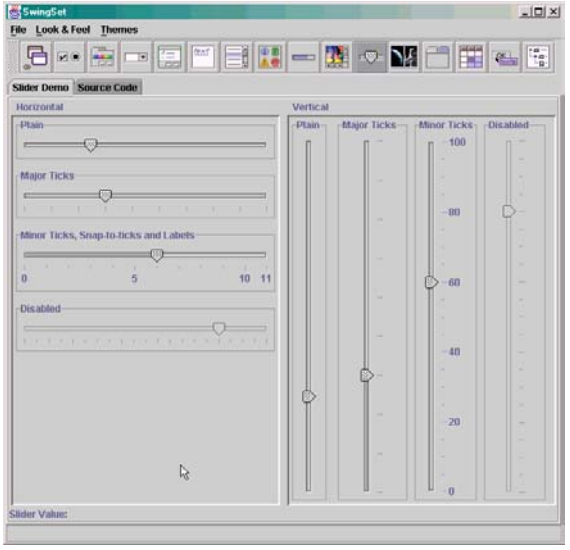
A Color Mixer



Exploring the Swing Documentation

- Don't try to understand all methods
- Focus on what you need to do your job
  - How do I construct a slider?
  - How can I get notified when the user has moved it?
  - How can I tell what the user has set it to? When you complete the basics, look again
- How about those “tick marks”?

The Swing Set Demo



Constructing a Slider

- `JSlider()`
  Has range (0,100)
- `JSlider(int min, int max, int value)`
  Can specify range and initial value
- `JSlider(BoundedRangeModel m)`
  appears to be some internal mechanism

Listening to Slider

- **Look for "addXxxListener":**
  `void addChangeListener(ChangeListener l)`
- **What is a change listener? It has a single method**
  `void stateChanged(ChangeEvent e)`
- **How can we tell new slider setting?**
  `int getValue()`
- **Plan: Add the same change listener to all three sliders**
- **Listener method reads all slider values and updates color**

File SliderTest.java

```
1 import javax.swing.JFrame;
2
3 public class SliderTest
4 {
5    public static void main(String[] args)
6    {
7       SliderFrame frame = new SliderFrame();
8       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9       frame.show();
10    }
11 }
12
```

File SliderFrame.java

```
1 import java.awt.BorderLayout;
2 import java.awt.Color;
3 import java.awt.Container;
4 import java.awt.Dimension;
5 import java.awt.GridLayout;
6 import java.awt.event.WindowAdapter;
7 import java.awt.event.WindowEvent;
8 import javax.swing.JFrame;
9 import javax.swing.JLabel;
10 import javax.swing.JPanel;
11 import javax.swing.JSlider;
12 import javax.swing.SwingConstants;
13 import javax.swing.event.ChangeListener;
14 import javax.swing.event.ChangeEvent;
15
16 class SliderFrame extends JFrame
17 {
```

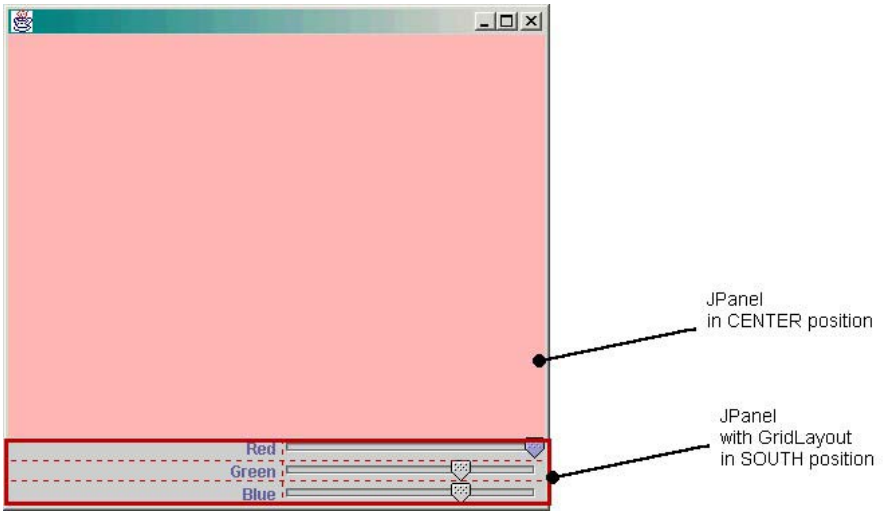
```
18  public SliderFrame()
19  {
20     colorPanel = new JPanel();
21     colorPanel.setPreferredSize(new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
22
23     getContentPane().add(colorPanel, BorderLayout.CENTER);
24     createControlPanel();
25     setSampleColor();
26     pack();
27  }
28
29  public void createControlPanel()
30  {
31     class ColorListener implements ChangeListener
32     {
33        public void stateChanged(ChangeEvent event)
34        {
35           setSampleColor();
36        }
37     }
```

```
38
39     ChangeListener listener = new ColorListener();
40
41     redSlider = new JSlider(0, 100, 100);
42     redSlider.addChangeListener(listener);
43
44     greenSlider = new JSlider(0, 100, 70);
45     greenSlider.addChangeListener(listener);
46
47     blueSlider = new JSlider(0, 100, 70);
48     blueSlider.addChangeListener(listener);
49
50     JPanel controlPanel = new JPanel();
51     controlPanel.setLayout(new GridLayout(3, 2));
52
53     controlPanel.add(new JLabel("Red",
54        SwingConstants.RIGHT));
55     controlPanel.add(redSlider);
56
57     controlPanel.add(new JLabel("Green",
```

```
58          SwingConstants.RIGHT));
59       controlPanel.add(greenSlider);
60
61       controlPanel.add(new JLabel("Blue",
62          SwingConstants.RIGHT));
63       controlPanel.add(blueSlider);
64
65       getContentPane().add(controlPanel, BorderLayout.SOUTH);
66    }
67
68
69    /**
70       Reads the slider values and sets the panel to
71       the selected color.
72    */
73    public void setSampleColor()
74    {  // read slider values
75
76       float red = 0.01F * redSlider.getValue();
77       float green = 0.01F * greenSlider.getValue();
```

```
78       float blue = 0.01F * blueSlider.getValue();
79
80       // set panel background to selected color
81
82       colorPanel.setBackground(new Color(red, green, blue));
83       colorPanel.repaint();
84    }
85
86    private JPanel colorPanel;
87    private JSlider redSlider;
88    private JSlider greenSlider;
89    private JSlider blueSlider;
90
91    private static final int PANEL_WIDTH = 300;
92    private static final int PANEL_HEIGHT = 300;
93 }
```

The Components of the Slider Frame



Classes of the Slider Test Program