Java Programming Tutorial

Programming Graphical User Interface (GUI) - Part 2

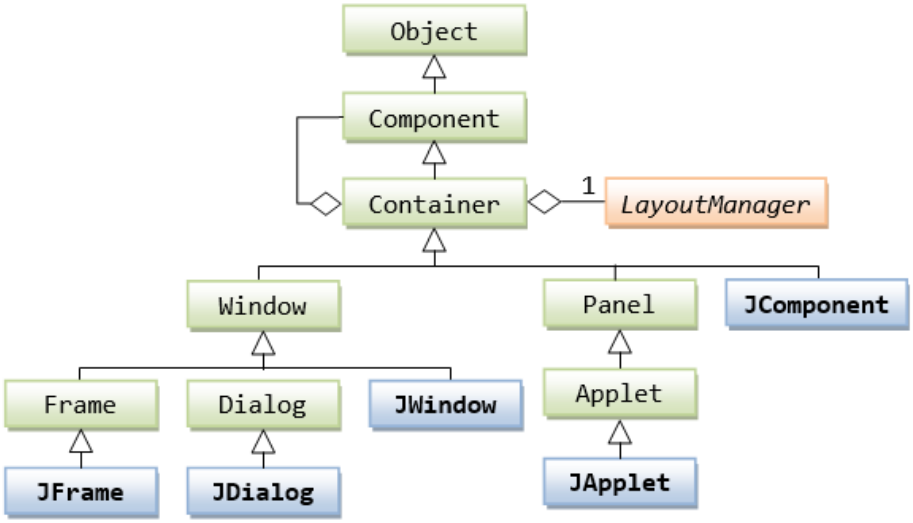
TABLE OF CONTENTS (HIDE)

1. More on Swing's JComponents
   - 1.1 ImageIcon
   - 1.2 Setting the Appearances and I
   - 1.3 Display Area, Border and Inset
   - 1.4 Positioning Your Application V
   - 1.5 Text Components: JTextFiel
   - 1.6 Buttons and ComboBox: JBut
   - 1.7 Menu-Bar: JMenuBar, JMenu, J
   - 1.8 JOptionPane: Interacting with
2. Pluggable Look and Feel
   - 2.1 Setting the Look and Feel
   - 2.2 Nimbus Look and Feel (JDK 1.
3. More on Layout Manager
   - 3.1 Key Points on Layout Manage
   - 3.2 Methods validate() and do
   - 3.3 add(), remove(), removeAll
   - 3.4 Component Orientation
   - 3.5 Absolute Positioning Without
4. More on Event-Handling
   - 4.1 java.util.EventObject
   - 4.2 ActionEvent & ActionListe
   - 4.3 Swing's Action
   - 4.4 WindowEvent & WindowListe
   - 4.5 KeyEvent & KeyListener/Ke
   - 4.6 MouseEvent & MouseListene
   - 4.7 MouseEvent & MouseMotionL

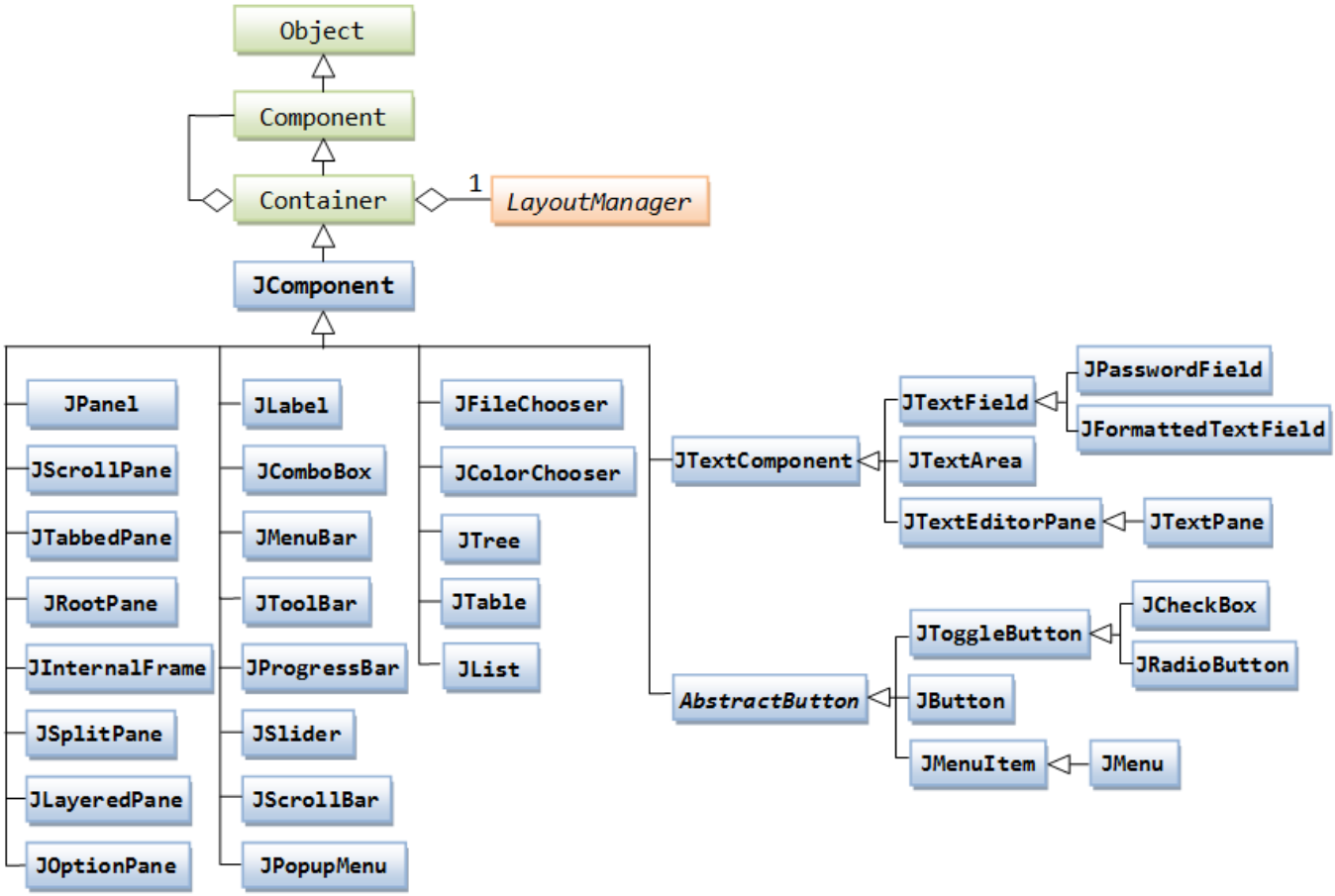
JDK demo includes a folder "jfc", which has many interesting demo on Swing and Java2D.

1. More on Swing's JComponents

The class hierarchy of Swing's top-level containers (JFrame, JDialog, JApplet) are as follows. These top-level Swing containers are heavyweight, that rely on the underlying windowing subsystem of the native platform.



The class hierarchy of Swing's JComponents is as follows. JComponent and its descendants are lightweight components.



1.1 ImageIcon

Many Swing's `JComponents` (such as `JLabel` and `JButton`) support a text label and an image icon. For example, the figure shows three buttons: one with text label, one with an image icon, and one with both text and icon.

The `javax.swing.ImageIcon` class models an image icon. An `ImageIcon` is a fixed-size picture, typically small, and mainly used for decorating GUI components. The `ImageIcon` class implements `javax.swing.Icon` interface, and hence, often upcasted and referenced as `Icon`.

To construct an `ImageIcon`, provide the image filename or URL. Image file type of GIF, PNG, JPG and BMP are supported. For example,

```
// Construct an ImageIcon from an image filename
String imgFilename = "images/duke.png";
      // Can use an absolute filename such as "c:/project/images/nought.gif"
ImageIcon iconDuke = new ImageIcon(imgFilename);

// OR
// Construct an ImageIcon via an image URL (in the form of file://path/filename)
ImageIcon iconDuke = null;
String imgFilename = "images/duke.png";
java.net.URL imgURL = getClass().getClassLoader().getResource(imgFilename);
      // Filename always relative to the root of the project (i.e., bin)
      // can access resource in a JAR file
if (imgURL != null) {
   iconDuke = new ImageIcon(imgURL);
} else {
   System.err.println("Couldn't find file: " + imgFilename);
}
```

Using URL is more flexible as it can access resources in a JAR file, and produces an error message if the file does not exist (which results in a `null` URL).

Many `JComponents` (such as `JLabel`, `JButton`) accepts an `ImageIcon` in its *constructor*, or via the `setIcon()` method. For example,

```
ImageIcon iconDuke = null;
String imgFilename = "images/duke.gif";  // relative to project root (or bin)
URL imgURL = getClass().getClassLoader().getResource(imgFilename);
if (imgURL != null) {
```

```
   iconDuke =  new ImageIcon(imgURL);
} else {
   System.err.println("Couldn't find file: " + imgFilename);
}

JLabel lbl = new JLabel("The Duke", iconDuke, JLabel.CENTER);
lbl.setBackground(Color.LIGHT_GRAY);
lbl.setOpaque(true);

Container cp = getContentPane();
cp.add(lbl);
```

An `ImageIcon` uses an `java.awt.Image` object to hold the image data. You can retrieve this `Image` object via the `ImageIcon`'s `getImage()` method. The `Image` object is used in the `drawImage()` method for custom drawing (which shall be discussed later).

1.2 Setting the Appearances and Properties of `JComponents`

Most of the Swing Components supports these features:

- Text and icon.
- Keyboard short-cut (called *mnemonics*), e.g., activated via the "Alt" key in Windows System.
- Tool tips: display when the mouse-pointer pauses on the component.
- Look and feel: customized appearance and user interaction for the operating platform.
- Localization: different languages for different locale.

All `JComponents` (such as `JPanel`, `JLabel`, `JTextField` and `JButton`) support these set methods to set their appearances and properties:

```
// javax.swing.JComponent
public void setBackground(Color bgColor)
   // Sets the background color of this component
public void setForeground(Color fgcolor)
   // Sets the foreground (text) color of this component
public void setFont(Font font)
   // Sets the font used by this component
public void setBorder(Border border)
   // Sets the border for this component
public void setPreferredSize(Dimension dim)
public void setMaximumSize(Dimension dim)
public void setMinimumSize(Dimension dim)
   // Sets the preferred, maximum or minimum size of this component.
public void setOpaque(boolean isOpaque)
   // If true (opaque), fill the background with background color;
   // otherwise, enable transparent background.
   // Most of the JComponents have default of true, except JLabel.
public void setToolTipText(String toolTipMsg)
   // Sets the tool-tip message, to be displayed when the mouse-pointer pauses over the component.
```

Swing's `JLabel` and buttons (`AbstractButton` subclasses): support both text and icon, which can be specified in the constructor or via the setters.

```
// javax.swing.JLabel, javax.swing.AbstractButton
public void setText(String strText)
   // Set the text
public void setIcon(Icon defaultIcon)
   // Set the button's default icon (you can have different icons for "pressed" and "disabled" states)
public void setHorizontalAlignment(int alignment)
   // Set the horizontal alignment of icon and text
   // SwingConstants.RIGHT, SwingConstants.LEFT, SwingConstants.CENTER
public void setVerticalAlignment(int alignment)
   // Set the vertical alignment of icon and text,
   // SwingConstants.TOP, SwingConstants.BOTTOM, SwingConstants.CENTER
public void setHorizontalTextPosition(int textPosition)
   // Set the horizontal text position relative to icon
   // SwingConstants.RIGHT, SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.LEADING, SwingConstants.TRAILING.
public void setVerticalTextPosition(int textPosition)
   // Set the vertical text position relative to icon
   // SwingConstants.TOP, SwingConstants.BOTTOM, SwingConstants.CENTER
```

`JTextField` supports:

```
// javax.swing.JTextField, javax.swing.JLabel, javax.swing.AbstractButton
public void setHorizontalAlignment(int alignment)
   // Set the text's horizontal alignment: JTextField.LEFT, JTextField.CENTER
```

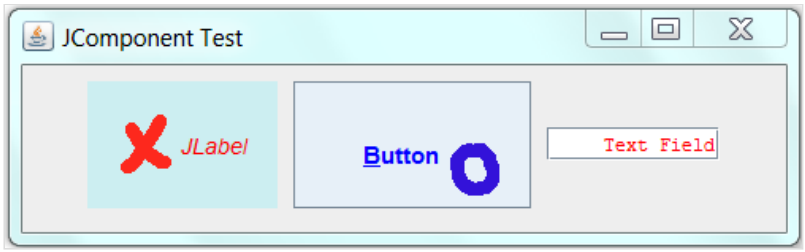
```
   //    JTextField.RIGHT, JTextField.LEADING, JTextField.TRAILING
   // No setVerticalAlignment as text field is single-line
```

Swing's buttons support mnemonic (to be triggered via keyboard short-cut with alt key).

```
// javax.swing.AbstractButton
public void setMnemonic(int mnemonic)
   // Set the keyboard mnemonic (i.e., the alt short-cut key).
   // Use KeyEvent.VK_xxx to specify the key.
```

Example

This example creates 3 `JComponents`: a `JLabel`, a `JTextField` and a `JButton`, and sets their appearances (background and foreground colors, font, preferred size and opacity). It also sets the horizontal text alignment for the `JTextField`.



Images:

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.net.URL;
4  import javax.swing.*;
5
6  /** Test setting Swing's JComponents properties and appearances */
7  @SuppressWarnings("serial")
8  public class SwingJComponentSetterTest extends JFrame {
9
10    // Image path relative to the project root (i.e., bin)
11    private String imgCrossFilename = "images/cross.gif";
12    private String imgNoughtFilename = "images/nought.gif";
13
14    /** Constructor to setup the GUI */
15    public SwingJComponentSetterTest() {
16
17       // Prepare ImageIcons to be used with JComponents
18       ImageIcon iconCross = null;
19       ImageIcon iconNought = null;
20       URL imgURL = getClass().getClassLoader().getResource(imgCrossFilename);
21       if (imgURL != null) {
22          iconCross = new ImageIcon(imgURL);
23       } else {
24          System.err.println("Couldn't find file: " + imgCrossFilename);
25       }
26       imgURL = getClass().getClassLoader().getResource(imgNoughtFilename);
27       if (imgURL != null) {
28          iconNought = new ImageIcon(imgURL);
29       } else {
30          System.err.println("Couldn't find file: " + imgNoughtFilename);
31       }
32
33       Container cp = getContentPane();
34       cp.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
35
36       // Create a JLabel with text and icon and set its appearances
37       JLabel label = new JLabel("JLabel", iconCross, SwingConstants.CENTER);
38       label.setFont(new Font(Font.DIALOG, Font.ITALIC, 14));
39       label.setOpaque(true);  // needed for JLabel to show the background color
40       label.setBackground(new Color(204, 238, 241));  // light blue
41       label.setForeground(Color.RED);                 // foreground text color
42       label.setPreferredSize(new Dimension(120, 80));
43       label.setToolTipText("This is a JLabel");  // Tool tip
44       cp.add(label);
45
46       // Create a JButton with text and icon and set its appearances
47       JButton button = new JButton(); // use setter to set text and icon
48       button.setText("Button");
49       button.setIcon(iconNought);
50       button.setVerticalAlignment(SwingConstants.BOTTOM);  // of text and icon
```

```
        button.setHorizontalAlignment(SwingConstants.RIGHT); // of text and icon
        button.setHorizontalTextPosition(SwingConstants.LEFT); // of text relative to icon
        button.setVerticalTextPosition(SwingConstants.TOP);    // of text relative to icon
        button.setFont(new Font(Font.SANS_SERIF, Font.BOLD, 15));
        button.setBackground(new Color(231, 240, 248));
        button.setForeground(Color.BLUE);
        button.setPreferredSize(new Dimension(150, 80));
        button.setToolTipText("This is a JButton");
        button.setMnemonic(KeyEvent.VK_B);  // can activate via Alt-B (buttons only)
        cp.add(button);

        // Create a JTextField with text and icon and set its appearances
        JTextField textField = new JTextField("Text Field", 15);
        textField.setFont(new Font(Font.DIALOG_INPUT, Font.PLAIN, 12));
        textField.setForeground(Color.RED);
        textField.setHorizontalAlignment(JTextField.RIGHT);  // Text alignment
        textField.setToolTipText("This is a JTextField");
        cp.add(textField);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("JComponent Test");
        setLocationRelativeTo(null); // center window on the screen
        setSize(500, 150);  // or pack()
        setVisible(true);

        // Print description of the JComponents via toString()
        System.out.println(label);
        System.out.println(button);
        System.out.println(textField);
    }

    /** The entry main() method */
    public static void main(String[] args) {
        // Run the GUI codes on Event-Dispatching thread for thread safety
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new SwingJComponentSetterTest(); // Let the constructor do the job
            }
        });
    }
}
```

```
javax.swing.JLabel[, 41, 10, 120x80, alignmentX=0.0, alignmentY=0.0, border=, flags=25165832,
      maximumSize=, minimumSize=, preferredSize=java.awt.Dimension[width=120,height=80],
      defaultIcon=file:.../cross.gif, disabledIcon=,
      horizontalAlignment=CENTER, horizontalTextPosition=TRAILING,
      iconTextGap=4, labelFor=, text=JLabel, verticalAlignment=CENTER, verticalTextPosition=CENTER]
javax.swing.JButton[, 171, 10, 150x80, alignmentX=0.0, alignmentY=0.5,
      border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@c5e2cf, flags=424,
      maximumSize=, minimumSize=, preferredSize=java.awt.Dimension[width=150,height=80],
      defaultIcon=file:/.../nought.gif, disabledIcon=, disabledSelectedIcon=,
      margin=javax.swing.plaf.InsetsUIResource[top=2,left=14,bottom=2,right=14],
      paintBorder=true, paintFocus=true, pressedIcon=, rolloverEnabled=true, rolloverIcon=,
      rolloverSelectedIcon=, selectedIcon=, text=Button, defaultCapable=true]
javax.swing.JTextField[, 331, 39, 109x21, layout=javax.swing.plaf.basic.BasicTextUI$UpdateHandler,
      alignmentX=0.0, alignmentY=0.0, border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@1991de1,
      flags=296, maximumSize=, minimumSize=, preferredSize=,
      caretColor=sun.swing.PrintColorUIResource[r=51,g=51,b=51],
      disabledTextColor=javax.swing.plaf.ColorUIResource[r=184,g=207,b=229], editable=true,
      margin=javax.swing.plaf.InsetsUIResource[top=0,left=0,bottom=0,right=0],
      selectedTextColor=sun.swing.PrintColorUIResource[r=51,g=51,b=51],
      selectionColor=javax.swing.plaf.ColorUIResource[r=184,g=207,b=229],
      columns=15, columnWidth=7, command=, horizontalAlignment=RIGHT]
```

The above clearly showed that there are many more properties that can be controlled.

1.3 Display Area, Border and Insets

Display Area

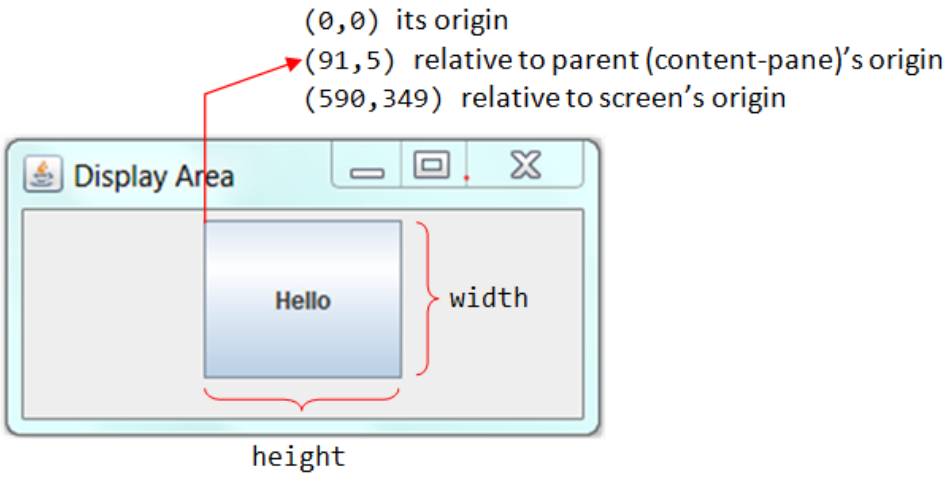
You can use the following get methods to get the dimensions of the display area of a `JComponent`. Each component maintains its own co-ordinates with origin (top-left corner) at (0, 0), width and height. You can also get the origin (x, y) relative to its parent or the screen.

```
public int getWidth()
public int getHeight()
public Dimension getSize()
   // Get the current width or height of the component, measured in pixels.
   // The getSize() returns width and height in a Dimension object.
public int getX()
public int getY()
public Point getLocation()
   // Get the component's current origin (x, y) relative to the
   //  parent's (e.g., JPanel or JFrame) upper-left corner, measured in pixels.
   // The getLocation() returns the (x, y) in a Point object.
public Point getLocationOnScreen()
   // Get the component's current origin (x, y) relative to the screen
```

For example:

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class TestSize {
5     public static void main(String[] args) {
6        JFrame frame = new JFrame("Display Area");
7        Container cp = frame.getContentPane();
8        cp.setLayout(new FlowLayout());
9        JButton btnHello = new JButton("Hello");
10       btnHello.setPreferredSize(new Dimension(100, 80));
11       cp.add(btnHello);
12
13       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14       frame.setSize(300, 150);  // or pack() the components
15       frame.setLocationRelativeTo(null);  // center the application window
16       frame.setVisible(true);             // show it
17
18       System.out.println(btnHello.getSize());
19       System.out.println(btnHello.getLocation());
20       System.out.println(btnHello.getLocationOnScreen());
21
22       System.out.println(cp.getSize());
23       System.out.println(cp.getLocation());
24       System.out.println(cp.getLocationOnScreen());
25
26       System.out.println(frame.getSize());
27       System.out.println(frame.getLocation());
28       System.out.println(frame.getLocationOnScreen());
29    }
30 }
```

```
java.awt.Dimension[width=100,height=80]   // JButton's getSize()
java.awt.Point[x=91,y=5]                  // JButton's getLocation()
java.awt.Point[x=590,y=349]               // JButton's getLocationOnScreen()

java.awt.Dimension[width=282,height=105]  // ContentPane's getSize()
java.awt.Point[x=0,y=0]                   // ContentPane's getLocation()
java.awt.Point[x=499,y=344]               // ContentPane's getLocationOnScreen()

java.awt.Dimension[width=300,height=150]  // JFrame's getSize()
java.awt.Point[x=490,y=308]               // JFrame's getLocation()
java.awt.Point[x=490,y=308]               // JFrame's getLocationOnScreen()
```



Border

Swing supports these border types (in package `javax.swing.border`), which can be applied to all `JComponents`:

1. `EmptyBorder`: empty, transparent border which takes up space but does no drawing.
2. `LineBorder`: line border of arbitrary thickness and of a single color.
3. `TitledBorder`: with the addition of a `String` title in a specified position and justification.
4. `StrokeBorder`: border of an arbitrary stroke.
5. `BevelBorder`: two-line bevel border.
6. `SoftBevelBorder`: raised or lowered bevel with softened corners.
7. `EtchedBorder`: etched-in or etched-out with highlight/shadow.
8. `MatteBorder`: matte-like border of either a solid color or a tiled icon.
9. `CompoundBorder`: compose two `Borders` - inner and outer.

To set a border to a `JComponent`, the easier way is to choose a `static` method from the `BorderFactory` class (instead of using the constructor of the `Border` class). For example,

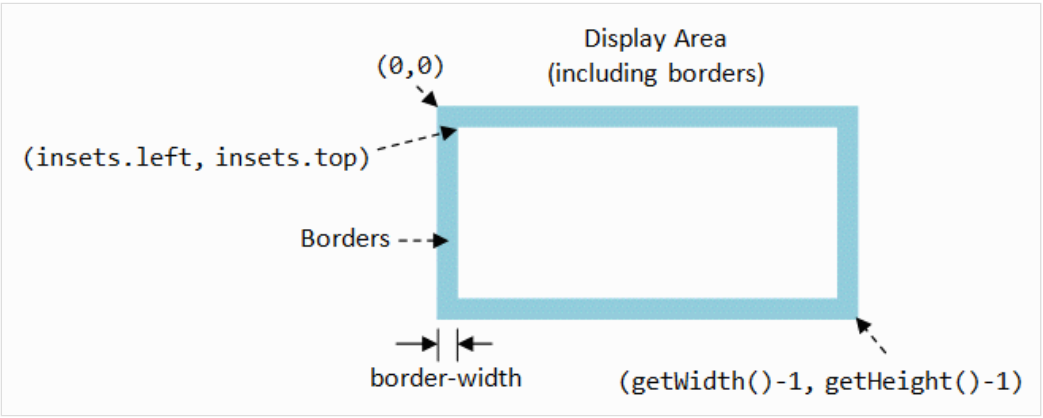
```
JPanel panel = new JPanel();
panel.setBorder(BorderFactory.createLineBorder(Color.CYAN, 15));
   // Set border to line-border with the given color and thickness
```

The `BorderFactory` class provides these `static` methods to create borders:

```
public static Border createLineBorder(Color color, [int thickness, boolean rounded])
   // Creates a line border with the specified color, thickness, and corner shape.
public static Border createEmptyBorder([int top, int left, int bottom, int right])
   // Creates an "empty" border with the given width of the top, left, bottom, and right sides.
public static TitledBorder createTitledBorder(String title)
   // Creates a new titled border with the specified title.
public static TitledBorder createTitledBorder(Border border, [String title,
      int titleJustification, int titlePosition, Font titleFont, Color titleColor])
   // Adds a title to an existing border, with the specified positioning, font and color.
public static CompoundBorder createCompoundBorder([Border outsideBorder, Border insideBorder])
   // Creates a compound border specifying the border objects to use for the outside and inside edges.
MORE: Refer to the API
```

Borders are included into the display area of a `JComponent` as illustrated.

To exclude the border, you could use the method `getInSets()` to retrieve the 4 borders in an `Insets` object (says `insets`), and use `insets.left`, `insets.right`, `insets.top`, and

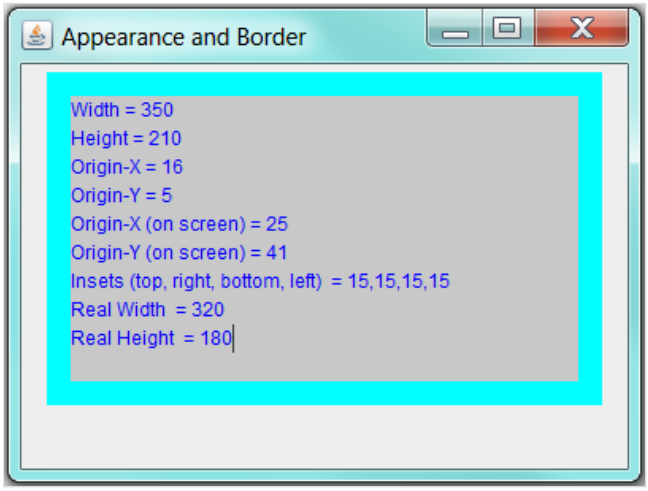


`insets.bottom` to retrieve the width of the 4 borders. For example,

```
// Continue from previous example
Insets insets = frame.getInsets();
System.out.println(insets);
   // java.awt.Insets[top=36,left=9,bottom=9,right=9]
int realWidth = frame.getWidth() - insets.left - insets.right;
int realHeight = frame.getHeight() - insets.top - insets.bottom;
System.out.println("real width = " + realWidth);    // real width = 282
System.out.println("real height = " + realHeight);  // real height = 105
```

Example

This example illustrates the display area, border and the various dimension.




```
1  import java.awt.*;
2  import javax.swing.*;
3
4  // A Swing application inherits from top-level container javax.swing.JFrame
5  public class TestDisplayAreaAndBorder extends JFrame {
6     /** Constructor to setup the GUI */
7     public TestDisplayAreaAndBorder() {
8        Container cp = getContentPane();
9        cp.setLayout(new FlowLayout());
10
11       JTextArea comp = new JTextArea(10, 25);  // row and columns
12       comp.setBackground(new Color(200, 200, 200));
13       comp.setForeground(Color.BLUE);
14       comp.setBorder(BorderFactory.createLineBorder(Color.CYAN, 15));
15             // set border to line-border with the given color and thickness
16       comp.setPreferredSize(new Dimension(350, 200));
17       cp.add(comp);
18
19       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20       setTitle("Appearance and Border");
21       setSize(400, 300);
22       setVisible(true);
23
24       StringBuffer msg = new StringBuffer();
25       msg.append("Width = " + comp.getWidth());
26       msg.append("\nHeight = " + comp.getHeight());
27       msg.append("\nOrigin-X = " + comp.getX());
28       msg.append("\nOrigin-Y = " + comp.getY());
29       msg.append("\nOrigin-X (on screen) = " + comp.getLocationOnScreen().x);
30       msg.append("\nOrigin-Y (on screen) = " + comp.getLocationOnScreen().y);
31
32       Insets insets = comp.getInsets();
33       msg.append("\nInsets (top, right, bottom, left)  = "
34          + insets.top + "," + insets.right + "," + insets.bottom + "," + insets.left);
35       msg.append("\nReal Width  = " + (comp.getWidth() - insets.left - insets.right));
36       msg.append("\nReal Height  = " + (comp.getHeight() - insets.top - insets.bottom));
37
38       comp.setText(msg.toString());
39    }
40
41    /** The entry main() method */
42    public static void main(String[] args) {
```

```
43          // Run the GUI codes on Event-Dispatching thread for thread safety
44          SwingUtilities.invokeLater(new Runnable() {
45             @Override
46             public void run() {
47                new TestDisplayAreaAndBorder(); // Let the constructor do the job
48             }
49          });
50       }
51    }
```

1.4 Positioning Your Application Window

You can position your main application window (`JFrame`), or top-level container, on the screen, via:

```
// Set methods (in java.awt.Window)
// (x, y) specifies the origin (top-left corner) of the window on the screen
public void setSize(int width, int height)
public void setLocation(int x, int y)
public void setBounds(int x, int y, int width, int height)
public void setSize(Dimension dim)
public void setLocation(Point origin)
public void setBounds(Rectangle r)   // JDK 1.6

// The associated get methods (in java.awt.Component) are:
public int getWidth()
public int getHeight()
public int getX()
public int getY()
public Dimension getSize()
public Point getLocation()
public Rectangle getBounds()
   // No setX(), setY(), setWidth(), setHeight()
```

You can get the screen size via `static` method `Toolkit.getDefaultToolkit().getScreenSize()`. For example,

```
Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();  // Get the screen dimension
int screenWidth = dim.width;
int screenHeight = dim.heigth;
```

You can also run your application in full-screen mode (with or without decorations such as title bar), instead of window-mode. Read "Swing How-To".

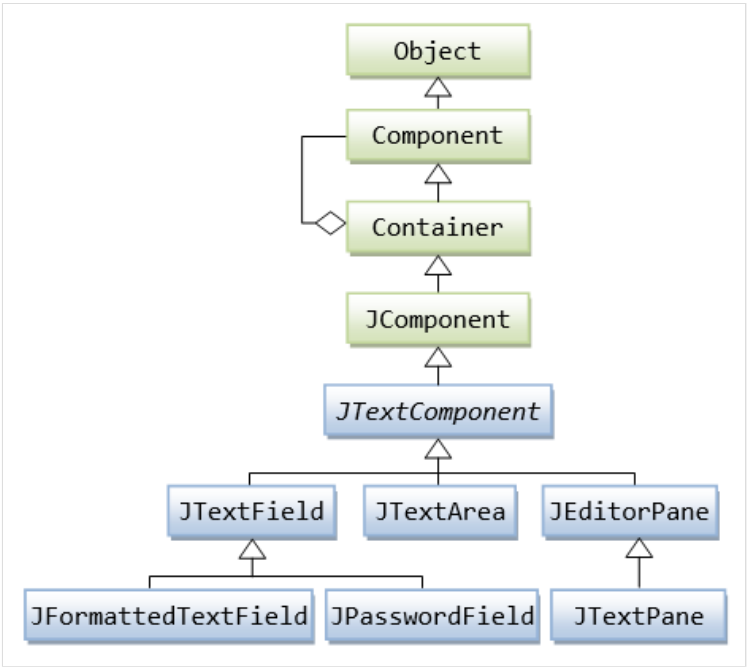
A quick way to center your application on the screen is to use `setLocationRelativeTo(null)`:

```
setSize(WINDOW_WIDTH, WINDOW_HEIGHT); // or pack() the components
setLocationRelativeTo(null);  // center the window on the screen
                              // shall be run after setSize()
setVisible(true);             // show it
```

1.5 Text Components: `JTextField`, `JTextArea`, `JEditorPane`

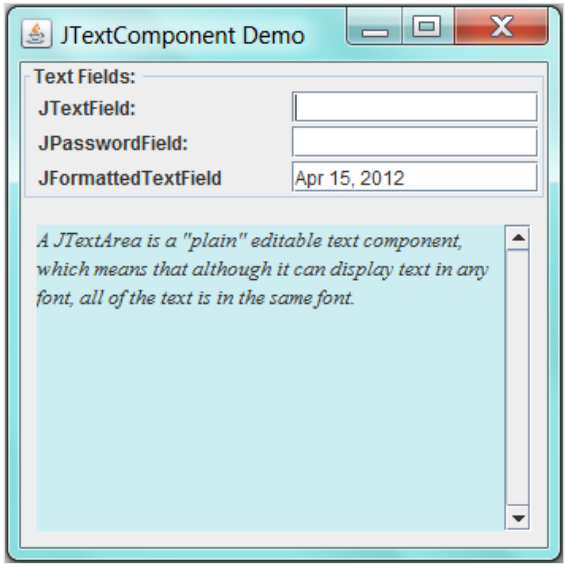
Swing provides 6 text components, as shown in the above class diagram. All text components extends from `JTextComponent`.

1. `JTextField`, `JPasswordField`, `JFormattedTextField`: For displaying only one line of editable text. Like buttons, they trigger `ActionEvent` when user hits the "enter" key.
2. `JTextArea`: Plain text area for displaying multiple lines of editable text, unformatted. All the texts are in the same font.
3. `JEditorPane`, `JTextPane`: A styled text area which can use more than one font. They support embedded images and embedded components. `JEditorPane` can load HMTL-formatted text from a URL.



Example: `JTextField`, `JPasswordField`, `JFormattedTextField`, and `JTextArea`

This example illustrates single-line `JTextField`, `JPasswordField`, `JFormattedField`, and multi-line `JTextArea` wrapped inside an `JScrollPane`.




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /** Test JTextField, JPasswordField, JFormattedTextField, JTextArea */
6  @SuppressWarnings("serial")
7  public class JTextComponentDemo extends JFrame {
8
9     // Private variables of the GUI components
10    JTextField tField;
11    JPasswordField pwField;
12    JTextArea tArea;
13    JFormattedTextField formattedField;
14
15    /** Constructor to set up all the GUI components */
16    public JTextComponentDemo() {
17       // JPanel for the text fields
18       JPanel tfPanel = new JPanel(new GridLayout(3, 2, 10, 2));
19       tfPanel.setBorder(BorderFactory.createTitledBorder("Text Fields: "));
20
21       // Regular text field (Row 1)
22       tfPanel.add(new JLabel("  JTextField: "));
23       tField = new JTextField(10);
24       tfPanel.add(tField);
25       tField.addActionListener(new ActionListener() {
```

```
         @Override
         public void actionPerformed(ActionEvent e) {
            tArea.append("\nYou have typed " + tField.getText());
         }
      });

      // Password field (Row 2)
      tfPanel.add(new JLabel("  JPasswordField: "));
      pwField = new JPasswordField(10);
      tfPanel.add(pwField);
      pwField.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            tArea.append("\nYou password is " + new String(pwField.getPassword()));
         }
      });

      // Formatted text field (Row 3)
      tfPanel.add(new JLabel("  JFormattedTextField"));
      formattedField = new JFormattedTextField(java.util.Calendar
            .getInstance().getTime());
      tfPanel.add(formattedField);

      // Create a JTextArea
      tArea = new JTextArea("A JTextArea is a \"plain\" editable text component, "
            + "which means that although it can display text "
            + "in any font, all of the text is in the same font.");
      tArea.setFont(new Font("Serif", Font.ITALIC, 13));
      tArea.setLineWrap(true);       // wrap line
      tArea.setWrapStyleWord(true);  // wrap line at word boundary
      tArea.setBackground(new Color(204, 238, 241)); // light blue
      // Wrap the JTextArea inside a JScrollPane
      JScrollPane tAreaScrollPane = new JScrollPane(tArea);
      tAreaScrollPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
      tAreaScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

      // Setup the content-pane of JFrame in BorderLayout
      Container cp = this.getContentPane();
      cp.setLayout(new BorderLayout(5, 5));
      cp.add(tfPanel, BorderLayout.NORTH);
      cp.add(tAreaScrollPane, BorderLayout.CENTER);

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setTitle("JTextComponent Demo");
      setSize(350, 350);
      setVisible(true);
   }

   /** The entry main() method */
   public static void main(String[] args) {
      // Run GUI codes in Event-Dispatching thread for thread safety
      SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
            new JTextComponentDemo();  // Let the constructor do the job
         }
      });
   }
}
```

`JPasswordField`

Use `getPassword()` to get the password entered in a `char[]`. [TODO: Security Issues]

`JFormattedTextField` (Advanced)

[TODO]

`JTextArea` wrapped in a `JScrollPane`

It is common to wrap a `JTextArea` inside a `JScrollPane`, so as to scroll the text area (horizontally or vertically). To do so, allocate a `JScrollPane` with the `JTextArea` as the argument.

```
JTextArea tArea = new JTextArea(...);  // Allocate JTextArea
JScrollPane tAreaScrollPane = new JScrollPane(tArea);  // Allocate JScrollPane which wraps the JTextArea
tAreaScrollPane.setVerticalScrollBarPolicy(...);       // Configure vertical scroll bar
tAreaScrollPane.setHorizontalScrollBarPolicy(...);     // Configure horizontal scroll bar
```

`JTextArea`'s Properties

You can replace the document, or append or insert more text.

```
public void append(String str)
   // Append the str to the end of the document
public void replaceRange(String str, int startPos, int endPos)
   // Replace with the str from startPos to endPos position
public void insert(String str, int pos)
   // Insert the str after the specified position
```

`JEditorPane` as HTML Browser

You can use `JEditorPane` to display an HTML document (but no CSS and JavaScript). Again, it is common to wrap a `JEditorPane` inside a `JScrollPane`. For example,

```
JEditorPane editorPane = new JEditorPane();  // allocate a JEditorPane
editorPane.setEditable(false);
try {
   // Form a URL and display the HTML page on the editor-pane
   URL url = new URL("http://www3.ntu.edu.sg/home/ehchua/programming/index.html");
   editorPane.setPage(url);
} catch (MalformedURLException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}
// Wrap the JEditorPane inside a JScrollPane
JScrollPane editorScrollPane = new JScrollPane(editorPane);
editorScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
setContentPane(editorScrollPane);
```

`JTextPane` (Advanced)

`JTextPane` is used for *formatted styled* text display; whereas `JTextArea` is used for *plain* text display. Although you can set the font and style for `JTextArea`, all the text is display in the same font. In `JTextPane`, you can set different fonts and styles for different sections of text.

You can use `JTextPane` for to develop an editor (such as NotePad or NotePad++) with different sections of text displayed in different fonts and colors.

Example: See Swing Tutorial's "Text Component Features". This is an excellent example that illustrates many features and concepts.

Text Component API: It provides many features, from cut and paste to changing the selected text. See Swing Tutorial's "Text Component API".

1.6 Buttons and ComboBox: `JButton`, `JCheckBox`, `JRadioButton`, `JComboBox`

Read: Swing Tutorial's "How to Use Buttons, Check Boxes, and Radio Buttons".

A user can click a button (or a menu item) to trigger a specific action. Swing supports mouse-less operations, where user could use a keyboard short-cut (called *mnemonic*) to activate the action.

Buttons and menu-items are inherited from `AbstractButton`, as shown in the class diagram.

You could place a text string (called button's label) as well as an icon on the button. You could use different icons for different button states: `defaultIcon`, `disabledIcon`, `pressedIcon`, `selectedIcon`, `rolloverIcon`, `disabledSelectedIcon`, `rolloverSelectedIcon`. The `defaultIcon` can be set via the constructor or `setIcon()` method. The other icons can be set via `setXxxIcon()` methods. You can set the alignment of text and button via `setHorizontalAlignment()` and `setVerticalAlignment()` methods. You can set the position of the text relative to the icon via `setHorizontalTextPosition()` and `setVerticalTextPosition()`.

Swing supports many type of buttons.

Command Buttons: `JButton`

Click to fires an `ActionEvent` to all its registered `ActionListeners`.

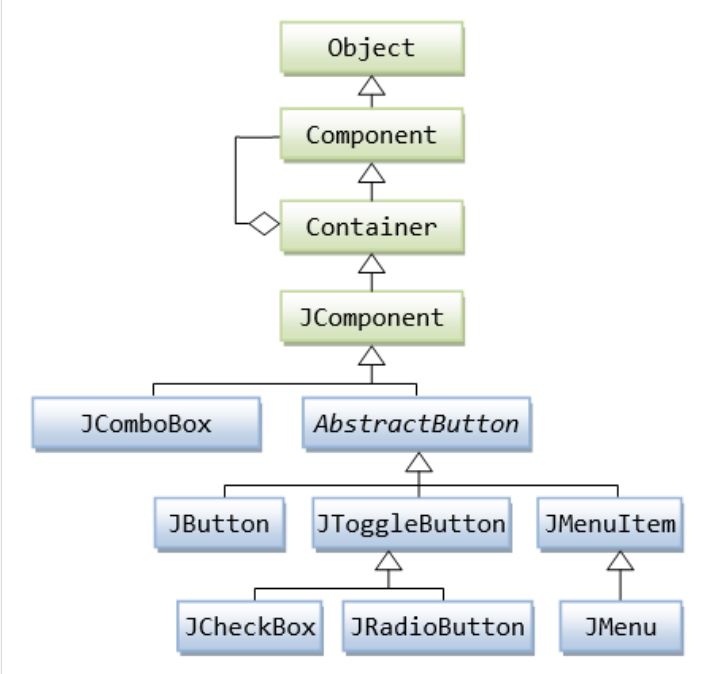
Toggle Buttons: `JRadioButton`, `JCheckBox`, `JToggleButton`

Toggle buttons are two-state buttons: `SELECTED` or `DESELECTED`.

For radio button, you can choose none or one among the group. `JRadioButtons` are typically added into a `ButtonGroup` to ensure exclusive selection. `JRadioButton` fires `ItemEvent` to its `ItemListeners`. It also fires `ActionEvent` to its `ActionListeners`. Both `ItemEvent` and `ActionEvent` can be used.

For checkboxes, you can choose none or more among the group. `JCheckBoxes` fire `ItemEvent` as well as `ActionEvent`. We typically use `ItemEvent` as we may need to distinguish between item-states of `SELECTED` and `DESELECTED`.

The `ItemListner` (of `ItemEvent`) declares one abstract method `itemStateChanged(ItemEvent e)`.

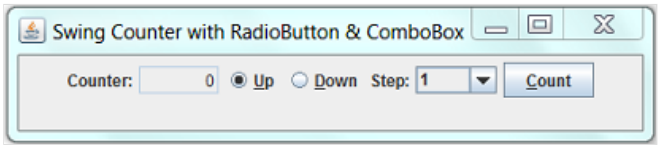


`JComboBox`

`JComboBox` can be used to provide a drop-down menu. It supports single-selection and multiple-selection. `JComboBox` receives a `Object` array (typically a `String` array), which provides the items in the drop-down list. `JComboBox` fires `ItemEvent`. We could use `JComboBox`'s `getSelectedIndex()` to get the index of the selected item; or `getSelectedItem()` to get the selected `Object`. `JComboBox` also fires `ActionEvent`.

Example on `JButton`, `JRadioButton` and `JComboBox`

In this example, we shall modify our counter application to include two radio buttons for specifying counting up/down, and a combo-box to select the count-step size.

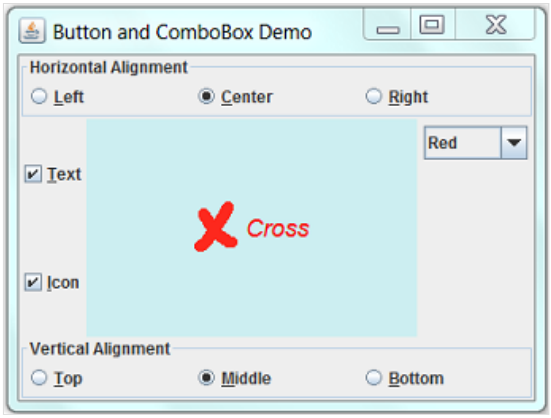



```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /** Counter with JRadioButton and JComboBox */
6  @SuppressWarnings("serial")
7  public class SwingCounterRadioCombo extends JFrame {
8     private JTextField tfCount;
9     private int count = 0;  // counter's value
10    private boolean countingUp = true; // counting up or down
11    private int step = 1;   // increment step size
12
13    /** Constructor to setup the UI */
14    public SwingCounterRadioCombo () {
15       Container cp = getContentPane();
16       cp.setLayout(new FlowLayout());
17
18       // Create JLabel and JTextField
19       cp.add(new JLabel("Counter:"));
20       tfCount = new JTextField("0", 5);
21       tfCount.setEditable(false);
22       tfCount.setHorizontalAlignment(JTextField.RIGHT);
23       cp.add(tfCount);
24
25       // Create JRadioButton for counting up and down
26       JRadioButton rbUp = new JRadioButton("Up", true);
27       rbUp.setMnemonic(KeyEvent.VK_U);
28       cp.add(rbUp);
29       rbUp.addActionListener(new ActionListener() {
30          @Override
31          public void actionPerformed(ActionEvent e) {
32             countingUp = true;
33          }
```

```
34          });
35          JRadioButton rbDown = new JRadioButton("Down", true);
36          rbDown.setMnemonic(KeyEvent.VK_D);
37          cp.add(rbDown);
38          rbDown.addActionListener(new ActionListener() {
39             @Override
40             public void actionPerformed(ActionEvent e) {
41                countingUp = false;
42             }
43          });
44          // Setup a ButtonGroup to ensure exclusive selection
45          ButtonGroup btnGp = new ButtonGroup();
46          btnGp.add(rbUp);
47          btnGp.add(rbDown);
48
49          // Create JComboBox for setting the count step size
50          add(new JLabel("Step:"));
51          final Integer[] steps = {1, 2, 3, 4, 5};  // auto-upcast
52          final JComboBox<Integer> comboCount = new JComboBox<Integer>(steps);
53          comboCount.setPreferredSize(new Dimension(60, 20));
54          cp.add(comboCount);
55          comboCount.addItemListener(new ItemListener() {
56             @Override
57             public void itemStateChanged(ItemEvent e) {
58                if (e.getStateChange() == ItemEvent.SELECTED) {
59                   step = (Integer)comboCount.getSelectedItem();
60                }
61             }
62          });
63
64          // Create JButton for "Count"
65          JButton btnCount = new JButton("Count");
66          btnCount.setMnemonic(KeyEvent.VK_C);
67          cp.add(btnCount);
68          btnCount.addActionListener(new ActionListener() {
69             @Override
70             public void actionPerformed(ActionEvent e) {
71                if (countingUp) {
72                   count += step;
73                } else {
74                   count -= step;
75                }
76                tfCount.setText(count + "");
77             }
78          });
79
80          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
81          setTitle("Swing Counter with RadioButton & ComboBox");
82          setSize(480, 100);
83          setVisible(true);
84       }
85
86       /** The entry main() method */
87       public static void main(String[] args) {
88          // Run the GUI codes in the Event-Dispatching thread for thread-safety
89          SwingUtilities.invokeLater(new Runnable() {
90             @Override
91             public void run() {
92                new SwingCounterRadioCombo(); // Let the constructor do the job
93             }
94          });
95       }
96    }
```

Example on `JRadioButton`, `JCheckBox` and `JComboBox`

In this example, we have two groups of radio buttons: one group to set the horizontal alignment of the `JLabel`, and processed via `ItemEvent`; another group sets the vertical alignment and processed via `ActionEvent`.




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.net.URL;
4  import javax.swing.*;
5
6  /** Test JRadioButton, JCheckBox and JComboBox */
7  @SuppressWarnings("serial")
8  public class ButtonComboBoxDemo extends JFrame {
9     // private variables of GUI components
10    private JLabel lblForTest;
11    private String imgCrossFilename = "images/cross.gif";
12    private String lblText = "Cross";
13    private Icon iconCross;
14
15    private JRadioButton rbLeft, rbCenter, rbRight, rbTop, rbMiddle, rbBottom;
16    private JCheckBox cbText, cbIcon;
17    private JComboBox<String> comboColor;
18
19    /** Constructor to setup the UI components */
20    public ButtonComboBoxDemo() {
21       // Create a JLabel with text and icon for manipulation
22       URL imgURL = getClass().getClassLoader().getResource(imgCrossFilename);
23       if (imgURL != null) {
24          iconCross = new ImageIcon(imgURL);
25       } else {
26          System.err.println("Couldn't find file: " + imgCrossFilename);
27       }
28       lblForTest = new JLabel(lblText, iconCross, SwingConstants.CENTER);
29       lblForTest.setOpaque(true);
30       lblForTest.setBackground(new Color(204, 238, 241));
31       lblForTest.setForeground(Color.RED);
32       lblForTest.setFont(new Font(Font.SANS_SERIF, Font.ITALIC, 18));
33
34       // Create radio buttons for setting horizontal alignment of the JLabel
35       rbLeft = new JRadioButton("Left");
36       rbLeft.setMnemonic(KeyEvent.VK_L);
37       rbCenter = new JRadioButton("Center", true);  // selected
38       rbCenter.setMnemonic(KeyEvent.VK_C);
39       rbRight = new JRadioButton("Right");
40       rbRight.setMnemonic(KeyEvent.VK_R);
41       // Put the radio buttons into a ButtonGroup to ensure exclusive selection
42       ButtonGroup btnGroupH = new ButtonGroup();
43       btnGroupH.add(rbLeft);
44       btnGroupH.add(rbRight);
45       btnGroupH.add(rbCenter);
46       // Set up a JPanel to hold all radio buttons
47       JPanel pnlRbtnH = new JPanel(new GridLayout(1, 0)); // single row
48       pnlRbtnH.add(rbLeft);
49       pnlRbtnH.add(rbCenter);
50       pnlRbtnH.add(rbRight);
51       pnlRbtnH.setBorder(BorderFactory.createTitledBorder("Horizontal Alignment"));
52
53       // A ItemListener for all Radio buttons
54       ItemListener listener = new ItemListener() {
55          @Override
56          public void itemStateChanged(ItemEvent e) {
57             if (e.getStateChange() == ItemEvent.SELECTED) {
```

```
            if (e.getSource() == rbLeft) {
               lblForTest.setHorizontalAlignment(SwingConstants.LEFT);
            } else if (e.getSource() == rbCenter) {
               lblForTest.setHorizontalAlignment(SwingConstants.CENTER);
            } else if (e.getSource() == rbRight) {
               lblForTest.setHorizontalAlignment(SwingConstants.RIGHT);
            }
         }
      };
      rbLeft.addItemListener(listener);
      rbCenter.addItemListener(listener);
      rbRight.addItemListener(listener);

      // Create radio buttons for setting vertical alignment of the JLabel
      rbTop = new JRadioButton("Top");
      rbTop.setMnemonic(KeyEvent.VK_T);
      rbMiddle = new JRadioButton("Middle", true);  // selected
      rbMiddle.setMnemonic(KeyEvent.VK_M);
      rbBottom = new JRadioButton("Bottom");
      rbBottom.setMnemonic(KeyEvent.VK_B);
      // Put the radio buttons into a ButtonGroup to ensure exclusive selection
      ButtonGroup btnGroupV = new ButtonGroup();
      btnGroupV.add(rbTop);
      btnGroupV.add(rbMiddle);
      btnGroupV.add(rbBottom);
      // Set up a JPanel to hold all radio buttons
      JPanel pnlRbtnV = new JPanel(new GridLayout(1, 0)); // single row
      pnlRbtnV.add(rbTop);
      pnlRbtnV.add(rbMiddle);
      pnlRbtnV.add(rbBottom);
      pnlRbtnV.setBorder(BorderFactory.createTitledBorder("Vertical Alignment"));

      // Radio buttons also fire ActionEvent
      rbTop.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            lblForTest.setVerticalAlignment(SwingConstants.TOP);
         }
      });
      rbMiddle.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            lblForTest.setVerticalAlignment(SwingConstants.CENTER);
         }
      });
      rbBottom.addActionListener(new ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            lblForTest.setVerticalAlignment(SwingConstants.BOTTOM);
         }
      });

      // Create checkboxes for selecting text, icon, or both, or none
      cbText = new JCheckBox("Text", true);  // selected
      cbText.setMnemonic(KeyEvent.VK_T);
      cbIcon = new JCheckBox("Icon", true);  // selected
      cbIcon.setMnemonic(KeyEvent.VK_I);
      cbIcon.setSelected(true);
      // Set up a JPanel to hold all checkboxes
      JPanel pnlCbox = new JPanel(new GridLayout(0, 1)); // single column
      pnlCbox.add(cbText);
      pnlCbox.add(cbIcon);
      // Checkboxes fire ItemEvent. Use an anonymous inner class as ItemListener
      cbText.addItemListener(new ItemListener() {
         @Override
         public void itemStateChanged(ItemEvent e) {
            // Need to handle both SELECTED and DESELECTED
            if (e.getStateChange() == ItemEvent.SELECTED) {
               lblForTest.setText(lblText);
            } else {
               lblForTest.setText("");
            }
```

```
            }
         });
         cbIcon.addItemListener(new ItemListener() {
            @Override
            public void itemStateChanged(ItemEvent e) {
               // Need to handle both SELECTED and DESELECTED
               if (e.getStateChange() == ItemEvent.SELECTED) {
                  lblForTest.setIcon(iconCross);
               } else {
                  lblForTest.setIcon(null);
               }
            }
         });

         // Create combobox (drop-down menu) for the foreground color of the JLabel
         String[] strColors = {"Red", "Blue", "Green",
                               "Cyan", "Magenta", "Yellow", "Black"};
         final Color[] colors = {Color.RED, Color.BLUE, Color.GREEN,
                         Color.CYAN, Color.MAGENTA, Color.YELLOW, Color.BLACK};
         comboColor = new JComboBox<String>(strColors);
         comboColor.addItemListener(new ItemListener() {
            @Override
            public void itemStateChanged(ItemEvent e) {
               if (e.getStateChange() == ItemEvent.SELECTED) {
                  lblForTest.setForeground(colors[comboColor.getSelectedIndex()]);
               }
            }
         });
         // Set up a JPanel for the combobox
         JPanel pnlCombo = new JPanel(new FlowLayout());
         pnlCombo.add(comboColor);

         // Set up the content-pane with BorderLayout and adds the panels
         Container cp = this.getContentPane();
         cp.setLayout(new BorderLayout());
         cp.add(lblForTest, BorderLayout.CENTER);
         cp.add(pnlRbtnH, BorderLayout.NORTH);
         cp.add(pnlRbtnV, BorderLayout.SOUTH);
         cp.add(pnlCbox, BorderLayout.WEST);
         cp.add(pnlCombo, BorderLayout.EAST);

         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         setTitle("Button and ComboBox Demo");
         setSize(400, 300);  // or pack() the components
         setLocationRelativeTo(null);
         setVisible(true);
      }

      /** The entry main() method */
      public static void main(String[] args) {
         // Run GUI codes in the Event-Dispatching thread for thread safety
         SwingUtilities.invokeLater(new Runnable() {
            public void run() {
               new ButtonComboBoxDemo();  // Let the constructor do the job
            }
         });
      }
   }
```

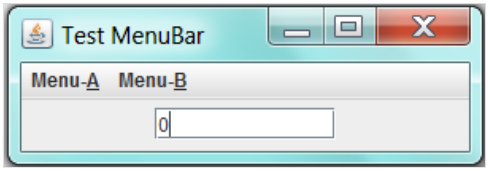
1.7 Menu-Bar: `JMenuBar`, `JMenu`, `JMenuItem`

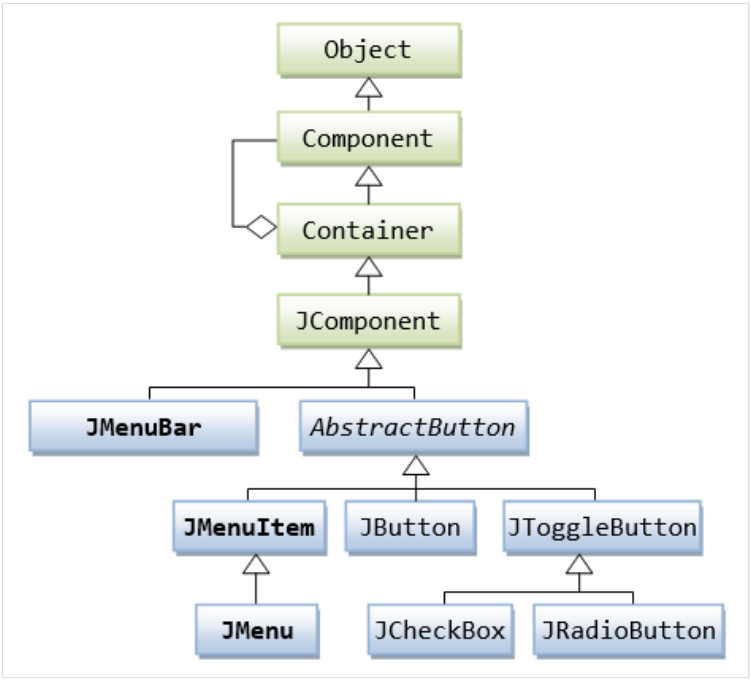
The menu-bar is at the same level as the content-pane (of the top-level container `JFrame`). It is set via the `JFrame`'s `setJMenuBar()` method (similar to `setContentPane()`).

To create a menu-bar, construct a `JMenuBar`. A menu-bar (`JMenuBar`) contains menu (`JMenu`). A menu contains menu-item (`JMenuItem`). `JMenuItem` is a subclass of `AbstractButton`, similar to `JButton`. `JMenuItem` fires `ActionEvent` upon activation to all its registered `ActionListener`.

Example

This menu-bar contains 2 menus (Menu-A and Menu-B). Menu-A contains 2 menu-items (Up and Down). Menu-B has 1 menu-item (Reset).






```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /** Testing menu-bar of JFrame */
6  public class TestJMenuBar extends JFrame {
7
8     JTextField display;
9     int count = 0;
10
11    /** Constructor to setup the GUI */
12    public TestJMenuBar() {
13       // A menu-bar contains menus. A menu contains menu-items (or sub-Menu)
14       JMenuBar menuBar;   // the menu-bar
15       JMenu menu;         // each menu in the menu-bar
16       JMenuItem menuItem; // an item in a menu
17
18       menuBar = new JMenuBar();
19
20       // First Menu
21       menu = new JMenu("Menu-A");
22       menu.setMnemonic(KeyEvent.VK_A);  // alt short-cut key
23       menuBar.add(menu);  // the menu-bar adds this menu
24
25       menuItem = new JMenuItem("Up", KeyEvent.VK_U);
26       menu.add(menuItem); // the menu adds this item
27       menuItem.addActionListener(new ActionListener() {
28          @Override
29          public void actionPerformed(ActionEvent e) {
30             ++count;
31             display.setText(count + "");
32          }
33       });
34
35       menuItem = new JMenuItem("Down", KeyEvent.VK_D);
36       menu.add(menuItem); // the menu adds this item
37       menuItem.addActionListener(new ActionListener() {
38          @Override
39          public void actionPerformed(ActionEvent e) {
40             --count;
41             display.setText(count + "");
42          }
43       });
44
45       // Second Menu
46       menu = new JMenu("Menu-B");
47       menu.setMnemonic(KeyEvent.VK_B);  // short-cut key
48       menuBar.add(menu);  // the menu bar adds this menu
```

```
49
50          menuItem = new JMenuItem("Reset", KeyEvent.VK_R);
51          menu.add(menuItem); // the menu adds this item
52          menuItem.addActionListener(new ActionListener() {
53             @Override
54             public void actionPerformed(ActionEvent e) {
55                count = 0;
56                display.setText(count + "");
57             }
58          });
59
60          setJMenuBar(menuBar);  // "this" JFrame sets its menu-bar
61
62          Container cp = getContentPane();
63          cp.setLayout(new FlowLayout());
64          display = new JTextField("0", 10);
65          cp.add(display);
66
67          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
68          setTitle("Test MenuBar");
69          setSize(300, 100);
70          setVisible(true);
71       }
72
73       /** The entry main() method */
74       public static void main(String[] args) {
75          // Run the GUI codes on the event-dispatching thread for thread safety
76          SwingUtilities.invokeLater(new Runnable() {
77             @Override
78             public void run() {
79                new TestJMenuBar(); // Let the constructor do the job
80             }
81          });
82       }
83    }
```

1.8 JOptionPane: Interacting with the User

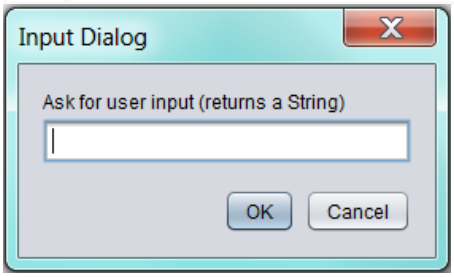
The `javax.swing.JOptionPane` provides standard *pre-built* diglog boxes to interact with user for both input and output. To create a dialog box, use one of the `static` methods `JOptionPane.showXxxDialog()`.

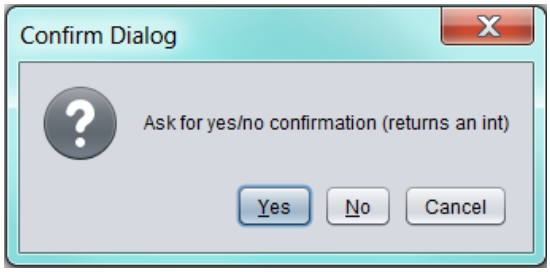
```
// Prompt for user input
public static String showInputDialog(Object message, [Object initialSelectionValue])
public static Object showInputDialog(Component parentComponent, Object message,
      [String title], [int messageType], [Icon icon], [Object[] options], [Object initialValue])

// Asks a confirming question (yes/no/cancel)
public static int showConfirmDialog(Component parentComponent, Object message,
      [String title], [int optionType], [int messageType], [Icon icon])

// Display a message
public static void showMessageDialog(Component parentComponent, Object message,
      [String title], [int messageType], [Icon icon])

// Support all features of the above three methods
public static int showOptionDialog(Component parentComponent, Object message,
      String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)

// Component parentComponent: parent of this dialog box, for poisitioning.
//    If null, centered on the screen.
// Object message: the message, typically a String.
// String title: the title for the dialog box
// int messageType: the style of the message,
//    JOptionPane.ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE.
// int optionType: the set of option buttons,
//    JOptionPane.DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, OK_CANCEL_OPTION
// Icon icon: a decorative icon, default depends on the messageType
// Object[] options, Object initialValue:
```

All these methods *block* the caller until the user's interaction is complete. Each of these methods also comes has a `showInternalXxxDialog()` version, which uses an internal frame to hold the dialog box.

Example: Input, Confirm and Message Dialogs

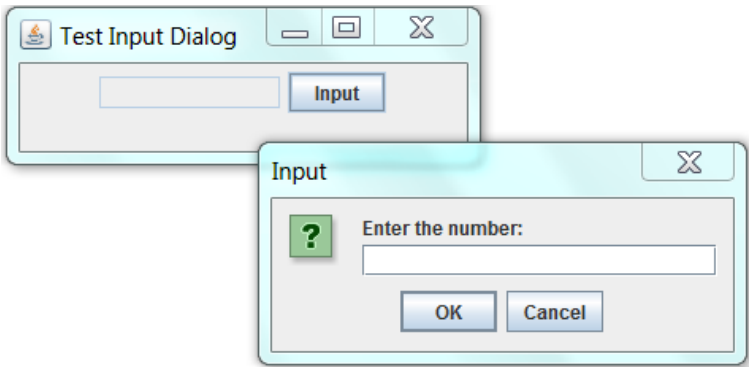
```
import javax.swing.*;
public class JOptionPaneTest {
   public static void main(String[] args) {
      // JOptionPane does not have to run under a Swing Application (extends JFrame).
      // It can run directly under main().
      String inStr = JOptionPane.showInputDialog(null, "Ask for user input (returns a String)",
            "Input Dialog", JOptionPane.PLAIN_MESSAGE);
      System.out.println("You have entered " + inStr);
      JOptionPane.showMessageDialog(null, "Display a message (returns void)!",
            "Message Dialog", JOptionPane.PLAIN_MESSAGE);
      int answer = JOptionPane.showConfirmDialog(null, "Ask for confirmation (returns an int)",
            "Confirm Dialog", JOptionPane.YES_NO_CANCEL_OPTION);
      switch (answer) {
         case JOptionPane.YES_OPTION:
            System.out.println("You clicked YES"); break;
         case JOptionPane.NO_OPTION:
            System.out.println("You clicked NO"); break;
         case JOptionPane.CANCEL_OPTION:
            System.out.println("You clicked Cancel"); break;
      }
   }
}
```





Take note that *input dialog* returns the `String` entered by the user; *confirm dialog* returns an int (`JOptionPane.YES_OPTION`, `NO_OPTION`, `CANCEL_OPTION`); *message dialog* returns `void`. Furthermore, you can use `JOptionPane` directly under `main()` to prompt user for input, similar to text-based input via `Scanner`.

Example: Prompting User for Input with Validation




```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class InputDialogWithValidation extends JFrame {
   JTextField tfDisplay;  // to display the number entered

   /** Constructor to setup the GUI components */
   public InputDialogWithValidation() {
      Container cp = getContentPane();
      cp.setLayout(new FlowLayout());

```

```
13          tfDisplay = new JTextField(10);
14          tfDisplay.setEditable(false);
15          cp.add(tfDisplay);
16
17          JButton btn = new JButton("Input");
18          cp.add(btn);
19          btn.addActionListener(new ActionListener() {
20             @Override
21             public void actionPerformed(ActionEvent e) {
22                boolean validInput = false;  // for input validation
23                int numberIn;
24                String inputStr = JOptionPane.showInputDialog("Enter a number [1-9]: ");
25                do {
26                   try {
27                      numberIn = Integer.parseInt(inputStr);
28                   } catch (NumberFormatException ex) {
29                      numberIn = -1;  // input triggered NumberFormatException, set to invalid
30                   }
31                   if (numberIn < 1 || numberIn > 9) {
32                      inputStr = JOptionPane.showInputDialog("Invalid numner! Enter a number [1-9]: ");
33                   } else {
34                      JOptionPane.showMessageDialog(null, "You have entered " + numberIn);
35                      validInput = true;
36                   }
37                } while (!validInput); // repeat if input is not valid
38                tfDisplay.setText(numberIn + "");
39             }
40          });
41          setDefaultCloseOperation(EXIT_ON_CLOSE);
42          setSize(300, 100);
43          setTitle("Test Input Dialog");
44          setVisible(true);
45       }
46
47       /** The "main" entry method */
48       public static void main(String[] args) {
49          // Run the GUI code on the Event-Dispatching Thread for thread safety
50          javax.swing.SwingUtilities.invokeLater(new Runnable() {
51             public void run() {
52                new InputDialogWithValidation(); // Let the constructor do the job
53             }
54          });
55       }
56    }
```

2. Pluggable Look and Feel

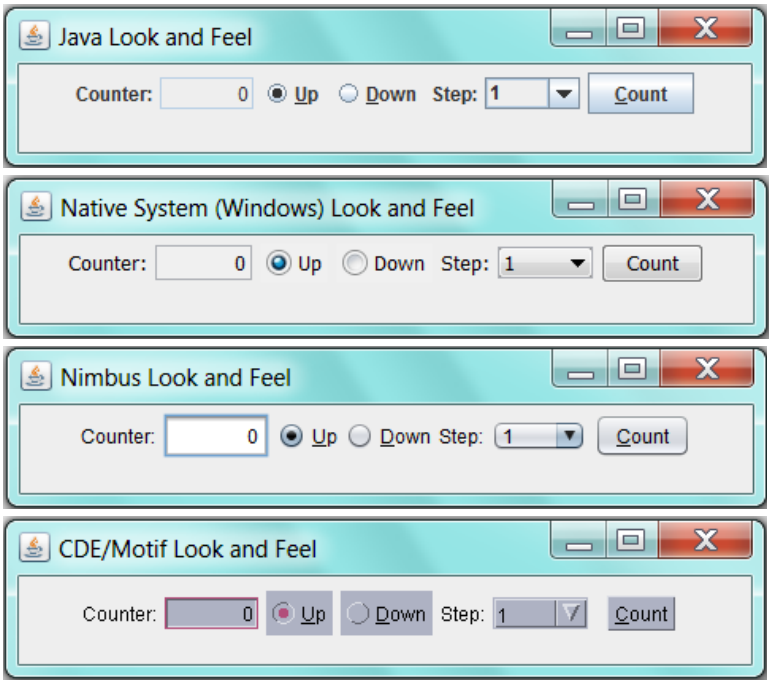
Swing supports the so-called "pluggable look and feel (plaf)" for its `JComponents`. The "look" refers to the appearance of the widgets (`JComponent`); while the "feel" refers to how the widgets behave (e.g., the behaviors of mouse-click for the various mouse-buttons). "Pluggable" refers the ability of changing the look and feel at runtime.

You can choose to use the default Java look and feel, the native system's look and feel (Windows, Linux, Mac), or the newer cross-platform Nimbus look and feel.

Pluggable look and feel is supported in Swing's components by separating the components into two classes: `JComponent` (in package `javax.swing`) and `ComponetUI` (in package `javax.swing.plaf`). The `ComponetUI`, called UI delegate, handles all aspects relating to look and feel. Nonetheless, you shall not interact with the UI delegate directly.

These look and feel are supported (in packages `javax.swing.plaf` and `javax.swing.plaf.*`):

1. Java Look and Feel: Also called `CrossPlatformLookAndFeel`, or Metal L&F. The default L&F which provides the same look and feel across all the platforms.
2. System Look and Feel: the L&F of the native system (e.g., Windows, Linux, Mac).
3. Nimbus Look and Feel: the newer cross-platform look and feel released in JDK 1.6 update 10.



The JFC demos (included in JDK demo) "SwingSet2" and "SwingSet3" show the various L&Fs.

2.1 Setting the Look and Feel

You need to set the Look and Feel as the first step in your GUI construction. There are a few ways to set the Look and Feel.

Via `UIManager.setLookAndFeel()`

You can use the `static` method `UIManager.setLookAndFeel(String` *`className`*`)` to set the look and feel.

- You can either use the `static` method `UIManager.getCrossPlatformLookAndFeelClassName()`, `UIManager.getSystemLookAndFeelClassName()` to get the classname string for Java F&F and Native System L&F; or
- Use the actual classname string such as "`javax.swing.plaf.metal.MetalLookAndFeel`" (for Java L&F), "`com.sun.java.swing.plaf.windows.WindowsLookAndFeel`" (Windows L&F), "`javax.swing.plaf.nimbus.NimbusLookAndFeel`" (Nimbus L&F) and "`com.sun.java.swing.plaf.motif.MotifLookAndFeel`" (Motif L&F).

For example,

```
// Set the UI manager (shall be the first step of the GUI construction)
try {
   // Set to cross-platform Java Look and Feel (also called "Metal")
   UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
} catch (UnsupportedLookAndFeelException e) {
   e.printStackTrace();
} catch (ClassNotFoundException e) {
   e.printStackTrace();
} catch (InstantiationException e) {
   e.printStackTrace();
} catch (IllegalAccessException e) {
   e.printStackTrace();
}
```

The alternative Look and Feel (under Windows System) are:

```
// Native System Look and Feel
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

// Cross-platform Java Look and Feel (also called "Metal")
UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

// Windows Look and Feel
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

// Cross-platform Nimbus Look and Feel (JDK 1.6u10)
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");

// Windows Classic Look and Feel
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel");
```

```
// CDE/Motif Look and Feel
UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
```

You can use `static` method `UIManager.getInstalledLookAndFeels()` to list all the installed L&F:

```
UIManager.LookAndFeelInfo[] lafs = UIManager.getInstalledLookAndFeels();
for (UIManager.LookAndFeelInfo laf : lafs) {
   System.out.println(laf);
}
```

```
javax.swing.UIManager$LookAndFeelInfo[Metal javax.swing.plaf.metal.MetalLookAndFeel]
javax.swing.UIManager$LookAndFeelInfo[Nimbus javax.swing.plaf.nimbus.NimbusLookAndFeel]
javax.swing.UIManager$LookAndFeelInfo[CDE/Motif com.sun.java.swing.plaf.motif.MotifLookAndFeel]
javax.swing.UIManager$LookAndFeelInfo[Windows com.sun.java.swing.plaf.windows.WindowsLookAndFeel]
javax.swing.UIManager$LookAndFeelInfo[Windows Classic com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel]
```

Via the Command Line Option "`swing.defaultlaf`"

For example,

```
// Set to Nimbus L&F
java -Dswing.defaultlaf=javax.swing.plaf.nimbus.NimbusLookAndFeel MySwingApp
// Set to Windows L&F
java -Dswing.defaultlaf=com.sun.java.swing.plaf.windows.WindowsLookAndFeel MySwingApp
```

Via the "`swing.properties`" file

Create a "`swing.properties`" file (placed under "`$JAVA_HOME/lib`" directory) with a option "`swing.defaultlaf`":

```
# Swing Look and Feel
swing.defaultlaf=javax.swing.plaf.nimbus.NimbusLookAndFeel
```

2.2 Nimbus Look and Feel (JDK 1.6u10)

Reference: Swing Tutorial's "Nimbus Look and Feel" @ http://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/nimbus.html.

Nimbus is a polished cross-platform look and feel introduced in the JDK 1.6 Update 10. The JFC demo "SwingSet3" (under the JDK demo) shows the Nimbus look and feel for the various Swing `JComponents`. "Nimbus uses Java 2D vector graphics to draw the user interface (UI), rather than static bitmaps, so the UI can be crisply rendered at any resolution. Nimbus is highly customizable. You can use the Nimbus look and feel as is, or you can skin (customize) the look with your own brand."

To enable Nimbus L&F:

1. Use `UIManager.setLookAndFeel()`:

   ```
   try {
      // Set to cross-platform Nimbus Look and Feel
      UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel"); // JDK 1.7
   } catch (Exception e) {
      e.printStackTrace();
   }

   // OR more robust codes
   try {
      // Check if Nimbus is supported and get its classname
      for (UIManager.LookAndFeelInfo lafInfo : UIManager.getInstalledLookAndFeels()) {
         if ("Nimbus".equals(lafInfo.getName())) {
            UIManager.setLookAndFeel(lafInfo.getClassName());
            break;
         }
      }
   } catch (Exception e) {
      try {
         // If Nimbus is not available, set to the default Java (metal) look and feel
         UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
      } catch (Exception e1) {
         e1.printStackTrace();
      }
   }
   ```

   Take note that the Nmibus package in JDK 1.6u10 is "`com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel`"; while in JDK 1.7, it is called "`javax.swing.plaf.nimbus.NimbusLookAndFeel`".

2. Use command-line option "`swing.defaultlaf`":

```
java -Dswing.defaultlaf=javax.swing.plaf.nimbus.NimbusLookAndFeel MySwingApp
```

3. Use a "`swing.properties`" file (under the "`$JAVA_HOME/lib`"):

```
# Set to Nimbus Look and Feel (System-wide)
swing.defaultlaf=javax.swing.plaf.nimbus.NimbusLookAndFeel
```

A Nimbus component can have 4 different sizes: large, regular, small and mini. You can choose the size via:

```
myButton.putClientProperty("JComponent.sizeVariant", "mini");
   // default is "regular", options are "large", "small" and "mini"
```

You can change the color theme via:

```
UIManager.put("nimbusBase", new Color(...));      // Base color
UIManager.put("nimbusBlueGrey", new Color(...));  // BlueGrey
UIManager.put("control", new Color(...));         // Control

// Set to Nimbus L&F
for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels() {
    if ("Nimbus".equals(info.getName())) {
        UIManager.setLookAndFeel(info.getClassName());
        break;
    }
}
......
```

You can customize the look and feel, which is beyond the scope of this article.

3. More on Layout Manager

Reference: Swing Tutorial's "Laying Out Components Within a Container" @ http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html.

3.1 Key Points on Layout Manager

Layout in Production

Use NetBeans' visual `GroupLayout` to layout the components in production; or `GridBagLayout` if you prefer to code yourself (why?). The rest of `LayoutManagers` (such as `FlowLayout`, `BorderLayout`, `GridLayout`) are meant for prototyping.

The two components you need to worry about layout are `JPanel` and the content-pane of the top-level containers (such as `JFrame`, `JApplet` and `JDialog`).

- `JPanel` defaults to `FlowLayout` (with alignment of `CENTER`, hgap and vgap of 5 pixels); or you can set the layout of a `JPanel` in its constructor.
- All content-panes default to `BorderLayout` (with hgap and vgap of 0). In `BorderLayout`, the `add(`*`aComponent`*`)` without specifying the zone adds the component to the `CENTER`. Second `add()` will override the first `add()`.

Absolute Positioning without `LayoutManager` shall be avoided, as it does not adjust well on screens with different resolutions; or when the window is resize.

Hints on sizes and alignments

You can provide hints on the minimum, preferred and maximum sizes of a component via `setMinimumSize()`, `setPreferredSize()` and `setMaximumSize()` methods. However, some layout managers ignore these requests, especially the maximum size. You can also do these by extending a subclass and overriding the `getXxxSize()` call-back methods.

The `setSize()` method must be issued at the correct point, or else it will not work (as it was overridden by another implicit `setSize()`). [TODO: Check]

Similarly, you can provide hints on horizontal and vertical alignments (of the edges) among components via `setAlignmentX()` and `setAlignmentY()` methods. `BoxLayout` honors them but other layout managers may ignore these hints. You can also extend a subclass and override the `getAlignmentX()` and `getAlignmentY()` call-back methods.

[TODO] `setSize()`, `pack()`, `validate()` and `invalidate()` for `Container`, `revalidate()` and `repaint()` for `Component`, `doLayout()`.

3.2 Methods `validate()` and `doLayout()`

If you change a property of a `LayoutManager`, such as `hgap` or `vgap` of `GridLayout`, you need to issue a `doLayout()` to force the `LayoutManager` to re-layout the components using the new property.

A container has only one `LayoutManager`. Nonetheless, you can change the `LayoutManager` via `setLayout(`*`newLayoutManager`*`)`. You need to follow with a `validate()` to ask the container to re-layout the components.

Code Example

This example creates 6 buttons, which are arranged in 3x2 and 2x3 `GridLayout` alternately upon clicking any button.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/** Changing the LayoutManager of the Container
     between 3*2 GridLayout and 2*3 GridLayout */
@SuppressWarnings("serial")
public class SetLayoutTest extends JFrame {
   private int rows = 3;
   private int cols = 2;
   private Container cp;  // content-pane of JFrame

   /** Constructor to setup the UI components */
   public SetLayoutTest() {
      cp = this.getContentPane();
      cp.setLayout(new GridLayout(rows, cols, 3, 3));

      // Create an instance of ActionListener to listen to all Buttons
      ButtonsListener listener = new ButtonsListener();

      // Create rows*cols Buttons and add to content-pane
      JButton[] buttons = new JButton[rows * cols];
      for (int i = 0; i < buttons.length; ++i) {
         buttons[i] = new JButton("Click [" + (i+1) + "]");
         cp.add(buttons[i]);
         buttons[i].addActionListener(listener);
      }

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setTitle("setLayout() Test");
      setSize(280, 150);
      setLocationRelativeTo(null);  // center the application window
      setVisible(true);
   }

   /** Inner class used as the ActionListener for the Buttons */
   private class ButtonsListener implements ActionListener {
      @Override
      public void actionPerformed(ActionEvent e) {
         // Swap rows and cols
         int temp = rows;
         rows = cols;
         cols = temp;

         // Set to new rows-by-cols GridLayout
         cp.setLayout(new GridLayout(rows, cols, 5, 5));
         cp.validate();  // ask LayoutManager to re-layout
      }
   }

   /** The entry main() method */
   public static void main(String[] args) {
      // Run GUI codes in the Event-Dispatching thread for thread safety
      SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            new SetLayoutTest();  // Let the constructor do the job
         }
      });
   }
}
```

Alternatively, you can also change the rows and columns of `GridLayout` via `setRows()` and `setColumns()` methods, and `doLayout()`. For example,

```
@Override
public void actionPerformed(ActionEvent e) {
   // Swap rows and cols
   int temp = rows;
   rows = cols;
   cols = temp;

   GridLayout layout = (GridLayout)cp.getLayout();
   layout.setRows(rows);
   layout.setColumns(cols);
   cp.doLayout();
}
```

3.3 add(), remove(), removeAll() Components from a Container

You can use *aContainer*.add(*aComponent*) to add a component into a container. You can also use *aContainer*.remove(*aComponent*) or *aContainer*.removeAll() to remove a component or all the components. You need to issue a validate() call to the container after adding or removing components.

Code Example

This example starts with 2 buttons: one to "add" a button and one to "remove" a button. The buttons are arranged in FlowLayout on the content-pane. For demonstration purpose, I remove all the buttons and re-add all the buttons.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /** Add or remove components from a container */
6  @SuppressWarnings("serial")
7  public class AddRemoveComponentsTest extends JFrame {
8     private int numButtons = 2; // number of buttons, init to 2 (1 add, 1 remove)
9     Container cp;               // content-pane of JFrame
10    JButton[] buttons;          // array of buttons
11    ButtonsListener listener;   // an ActionListener instance for all buttons
12
13    /** Constructor to setup the UI components */
14    public AddRemoveComponentsTest() {
15       cp = getContentPane();
16
17       // Create an instance of ActionListener to listen to all Buttons
18       listener = new ButtonsListener();
19       // Call helper method to create numButtons buttons
20       createButtons();
21
22       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23       setTitle("Add/Remove Components Test");
24       setSize(400, 150);
25       setLocationRelativeTo(null);
26       setVisible(true);
27    }
28
29    /** Create numButtons buttons on content-pane with FlowLayout */
30    private void createButtons() {
31       // For demonstration, all the components are removed, instead of
32       //  add or remove selected component.
33       cp.removeAll();
34       cp.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));
35
36       // Create buttons
37       buttons = new JButton[numButtons];  // allocate array
38       int i = 0;
39       // Create (numButtons-1) "Add" buttons, but minimum one
40       do {
41          buttons[i] = new JButton("Add");
42          cp.add(buttons[i]);
43          buttons[i].addActionListener(listener);
44          ++i;
45       } while (i < numButtons - 1);
46       // Create a "Remove" button if numButtons > 1
47       if (i == numButtons - 1) {
48          buttons[i] = new JButton("Remove");
```

```
            cp.add(buttons[i]);
            buttons[i].addActionListener(listener);
         }
         cp.validate(); // needed after adding/removing component
         repaint();     // needed to cleanup dirty background
      }

      /** Inner class used as the ActionListener for the Buttons */
      private class ButtonsListener implements ActionListener {
         @Override
         public void actionPerformed(ActionEvent e) {
            // adjust the number of buttons
            if (e.getActionCommand().equals("Add")) {
               ++numButtons;
            } else {
               if (numButtons >= 2) {
                  --numButtons;
               }
            }
            // Call helper method to create numButtons buttons
            createButtons();
         }
      }

      /** The entry main() method */
      public static void main(String[] args) {
         // Run GUI codes in the Event-Dispatching thread for thread safety
         SwingUtilities.invokeLater(new Runnable() {
            public void run() {
               new AddRemoveComponentsTest();  // Let the constructor do the job
            }
         });
      }
   }
```

3.4 Component Orientation

Most languages from written form left-to-right, but some otherwise. You can set the orientation on `Component` via:

```
// java.awt.Component
public void setComponentOrientation(ComponentOrientation o)
   // ComponentOrientation.LEFT_TO_RIGHT or RIGHT_TO_LEFT.
```

Since JDK 1.4, layout managers, such as `FlowLayout` and `BorderLayout`, can layout components according to component-orientation of the container. Some new terms were introduced. For example, in `BorderLayout`, instead of using `EAST`, `WEST`, `NORHT`, `SOUTH` (which are absolute), the term `LINE_START`, `LINE_END`, `PAGE_START`, `PAGE_END` were added which can adjust itself according to the component orientation. `LINE_START` is the same as `WEST`, if the component orientation is `LEFT_TO_RIGHT`. On the other hand, it is `EAST`, if the component orientation is `RIGHT_TO_LEFT`. Similarly, in `FlowLayout`'s alignment, `LEADING` and `TRAILING` were added in place of `LEFT` and `RIGHT`.

Code Example

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/** BorderLayout Demo */
@SuppressWarnings("serial")
public class BorderLayoutTest extends JFrame {
   public static final String TITLE = "BorderLayout Demo";

   // Private variables of GUI components
   private Container cp;  // content-pane of this JFrame
   private JButton btnNorth, btnSouth, btnCenter, btnEast, btnWest;
   private boolean leftToRight = true;

   /** Constructor to setup the UI components */
   public BorderLayoutTest() {
      cp = this.getContentPane();
      btnNorth = new JButton("PAGE_START [HIDE]");
      btnSouth = new JButton("PAGE_END [HIDE]");
      btnWest = new JButton("LINE_START [HIDE]");
```

```
21          btnEast = new JButton("LINE_END [HIDE]");
22          btnCenter = new JButton("CENTER [SHOW ALL, CHANGE ORIENTATION]");
23          btnCenter.setPreferredSize(new Dimension(300, 100)); // set size for CENTER
24
25          ActionListener listener = new ButtonListener();
26          btnNorth.addActionListener(listener);
27          btnSouth.addActionListener(listener);
28          btnEast.addActionListener(listener);
29          btnWest.addActionListener(listener);
30          btnCenter.addActionListener(listener);
31
32          // Add all buttons to the content-pane
33          addButtons();
34
35          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // exit if close button clicked
36          setTitle(TITLE); // "this" JFrame sets title
37          pack();  // "this" JFrame packs all its components
38          setLocationRelativeTo(null);  // center the application window
39          setVisible(true);             // show it
40       }
41
42       /** Helper method to add all buttons to the content-pane */
43       private void addButtons() {
44          cp.removeAll();
45          cp.setComponentOrientation(leftToRight ?
46                ComponentOrientation.LEFT_TO_RIGHT : ComponentOrientation.RIGHT_TO_LEFT);
47          cp.add(btnNorth, BorderLayout.PAGE_START);
48          cp.add(btnSouth, BorderLayout.PAGE_END);
49          cp.add(btnWest, BorderLayout.LINE_START);
50          cp.add(btnEast, BorderLayout.LINE_END);
51          cp.add(btnCenter, BorderLayout.CENTER);
52          cp.validate();
53       }
54
55       /** Inner class used as ActionListener for all buttons */
56       private class ButtonListener implements ActionListener {
57          @Override
58          public void actionPerformed(ActionEvent evt) {
59             JButton source = (JButton)evt.getSource();
60             if (source == btnCenter) {
61                leftToRight = !leftToRight;  // toggle
62                addButtons();
63             } else {
64                cp.remove(source);
65                cp.validate();
66             }
67          }
68       }
69
70       /** The entry main() method */
71       public static void main(String[] args) {
72          // Run GUI codes in the Event-Dispatching thread for thread safety
73          SwingUtilities.invokeLater(new Runnable() {
74             @Override
75             public void run() {
76                new BorderLayoutTest();  // Let the constructor do the job
77             }
78          });
79       }
80    }
```

3.5 Absolute Positioning Without a Layout Manager

You could use absolute position instead of a layout manager (such as `FlowLayout` or `BorderLayout`) by invoking method `setLayout(null)`. You can then position you components using the method `setBounds(int xTopLeft, int yTopLeft, int width, int height)`. For example:

```
import java.awt.*;
import javax.swing.*;

/** Test Absolute Positioning */
public class CGAbsolutePositioning extends JFrame {
   /** Constructor to setup the GUI components */
   public CGAbsolutePositioning() {
      Container cp = getContentPane();
      cp.setLayout(null); // disable layout manager - use absolute layout

      JPanel p1 = new JPanel();
      p1.setBounds(30, 30, 100, 100);
      p1.setBackground(Color.RED);
      cp.add(p1);

      JPanel p2 = new JPanel();
      p2.setBounds(150, 50, 120, 80);
      p2.setBackground(Color.BLUE);
      cp.add(p2);

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setTitle("Absolute Positioning Demo");
      setSize(400, 200);
      setVisible(true);
   }

   /** The entry main method */
   public static void main(String[] args) {
      // Run GUI codes in Event-Dispatching thread for thread safety
      SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
            new CGAbsolutePositioning(); // Let the constructor do the job
         }
      });
   }
}
```

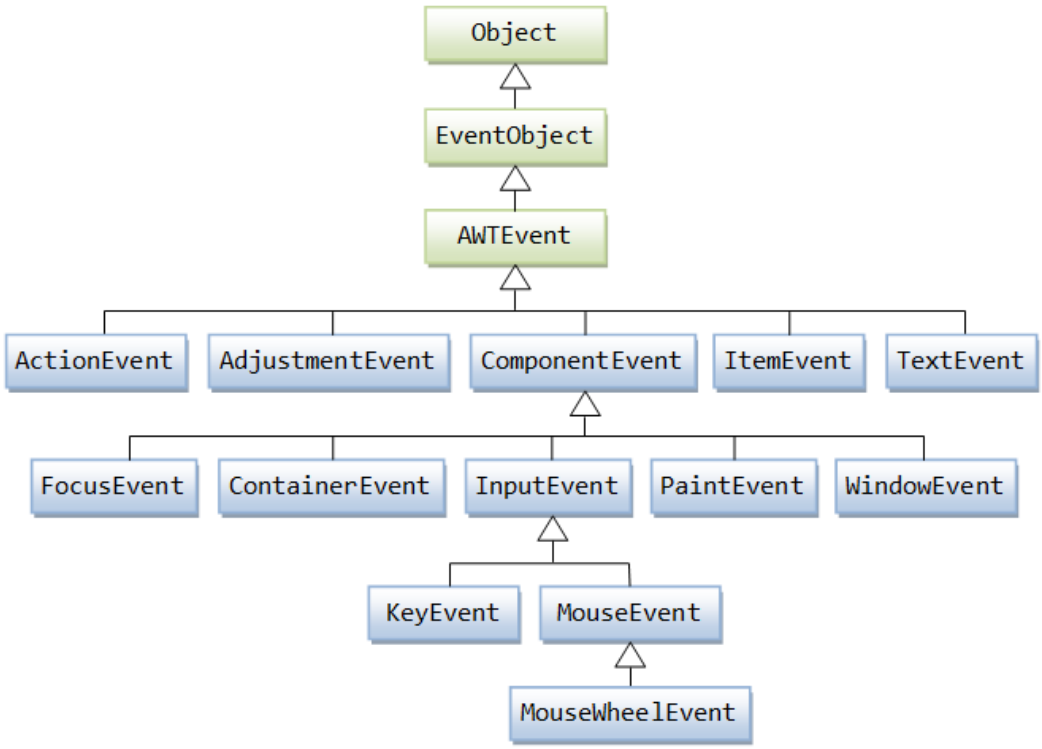
4. More on Event-Handling

Both AWT and Swing applications uses the AWT event-handling classes (in package `java.awt.event`). Swing added a few new event handling classes (in package `javax.swing.event`), but they are not frequently used.

AWT GUI Components (such as `Button`, `TextField`, and `Window`) can trigger an `AWTEvent` upon user's activation.

User Action	Event Triggered	Event Listener interface
Click a `Button`, `JButton`	`ActionEvent`	`ActionListener`
Open, iconify, close `Frame`, `JFrame`	`WindowEvent`	`WindowListener`
Click a `Component`, `JComponent`	`MouseEvent`	`MouseListener`
Change texts in a `TextField`, `JTextField`	`TextEvent`	`TextListener`
Type a key	`KeyEvent`	`KeyListener`
Click/Select an item in a `Choice`, `JCheckbox`, `JRadioButton`, `JComboBox`	`ItemEvent`, `ActionEvent`	`ItemListener`, `ActionListener`

The subclasses of `AWTEvent` are as follows:



4.1 java.util.EventObject

All event objects extends `java.util.EventObject`, which takes the source object in this constructor, and provides a `getSource()` method.

```
// Constructor
public EventObject(Object source)

// Method
public Object getSource()
```

Take note that the constructor takes an `Object`; and `getSource()` returns an instance of type `Object`. You may need to downcast it back to its original type.

4.2 ActionEvent & ActionListener

An `ActionEvent` is fired, when an action has been performed by the user. For examples, when the user clicks a button, chooses a menu item, presses enter key in a text field. The associated `ActionListener` interface declares only one abstract method, as follows:

```
public interface ActionListener extends java.util.EventListener {
   public void actionPerformed(ActionEvent evt); // called-back when an action has been performed
}
```

From the `ActionEvent` argument `evt`, you may use `evt.getActionCommand()` to get a `String` related to this event, for example, the button's label, the `String` entered into the textfield. This is particularly useful if the same `ActionEvent` handler is used to handle multiple source objects (e,g., buttons or textfields), for identifying the source object that triggers this `ActionEvent`.

4.3 Swing's Action

Read Swing Tutorial's "How to Use Actions".

A `javas.swing.Action` is a `ActionEvent` listener. If two or more components (e.g., a menu item and a button) perform the same function in response to an `ActionEvent`, you can use an `Action` object to specify both the *state* and *functionality* of the components (whereas `actionPerformed()` defines only the function). For example, you can specify the *states* such as the text, icon, shortcut key, tool-tip text, for all the source components.

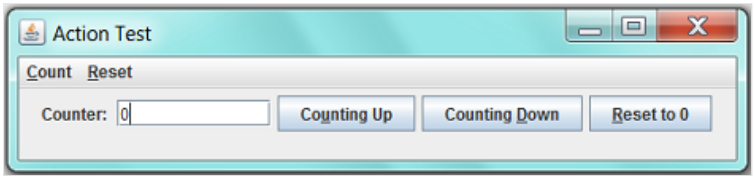
You can attach an `Action` object to a component via *`aComponent`*`.setAction(`*`anAction`*`)` method:

1. The component's state (e.g., text, icon) is updated to match the state of the `Action`.
2. The component adds the `Action` object as an `ActionEvent` listener.
3. If the state of the `Action` changes, the component's state is updated to match the `Action`.

To create an `Action` object, you extend `AbstractAction` to provide the state and implement the `actionPerformed()` method to response to the `ActionEvent`.

Example

In this example, a menu-item and a button share the same `Action`. The `Action` object specifies the states (text, tooltip's text, and a mnemonic alt short-cut key) and override the `actionPerformed()`. Take note that the label on the buttons are updated to match the `Action`'s names.




```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/**
 * Test Actions which are ActionListeners that can be applied to more than one sources.
 * An action can specify the state and functionality of an ActionListener.
 */
@SuppressWarnings("serial")
public class TestAction extends JFrame {
   private JTextField tfCount;
   private int count;

   /** Constructor to setup the GUI */
   public TestAction() {
      // Create the Actions shared by the button and menu-item
      Action countUpAction = new CountUpAction("Count Up",
            "To count up", new Integer(KeyEvent.VK_U));
      Action countDownAction = new CountDownAction("Count Down",
            "To count down", new Integer(KeyEvent.VK_D));
      Action resetAction = new ResetAction("Reset",
            "To reset to zero", new Integer(KeyEvent.VK_R));

      Container cp = getContentPane();
      cp.setLayout(new FlowLayout());

      // Create buttons
      cp.add(new JLabel("Counter: "));
      tfCount = new JTextField("0", 8);
      tfCount.setHorizontalAlignment(JTextField.RIGHT);
      cp.add(tfCount);
      JButton btnCountUp = new JButton();
      cp.add(btnCountUp);
      JButton btnCountDown = new JButton();
      cp.add(btnCountDown);
      JButton btnReset = new JButton();
      cp.add(btnReset);
      // Set actions for buttons
      btnCountUp.setAction(countUpAction);
      btnCountDown.setAction(countDownAction);
      btnReset.setAction(resetAction);

      // Create menu-bar
      JMenuBar menuBar = new JMenuBar();
      JMenu menu;
      JMenuItem menuItem;;

      // Create the first menu
      menu = new JMenu("Count");
      menu.setMnemonic(KeyEvent.VK_C);
      menuItem = new JMenuItem(countUpAction);   // allocate menu-item and set action
      menu.add(menuItem);
      menuItem = new JMenuItem(countDownAction); // allocate menu-item and set action
      menu.add(menuItem);
      menuBar.add(menu);

      // Create the second menu
      menu = new JMenu("Reset");
      menu.setMnemonic(KeyEvent.VK_R);
      menuItem = new JMenuItem(resetAction);  // allocate menu-item and set action
      menu.add(menuItem);
```

```
61          menuBar.add(menu);
62
63          setJMenuBar(menuBar); // "this" JFrame sets menu-bar
64          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
65          setTitle("Action Test");
66          setSize(550, 120);
67          setVisible(true);
68       }
69
70       /**
71        * Action inner classes
72        */
73       public class CountUpAction extends AbstractAction {
74          /** Constructor */
75          public CountUpAction(String name, String shortDesc, Integer mnemonic) {
76             super(name);
77             putValue(SHORT_DESCRIPTION, shortDesc);
78             putValue(MNEMONIC_KEY, mnemonic);
79          }
80
81          @Override
82          public void actionPerformed(ActionEvent e) {
83             ++count;
84             tfCount.setText(count + "");
85          }
86       }
87
88       public class CountDownAction extends AbstractAction {
89          /** Constructor */
90          public CountDownAction(String name, String shortDesc, Integer mnemonic) {
91             super(name);
92             putValue(SHORT_DESCRIPTION, shortDesc);
93             putValue(MNEMONIC_KEY, mnemonic);
94          }
95
96          @Override
97          public void actionPerformed(ActionEvent e) {
98             --count;
99             tfCount.setText(count + "");
100         }
101      }
102
103      public class ResetAction extends AbstractAction {
104         /** Constructor */
105         public ResetAction(String name, String shortDesc, Integer mnemonic) {
106            super(name);
107            putValue(SHORT_DESCRIPTION, shortDesc);
108            putValue(MNEMONIC_KEY, mnemonic);
109         }
110
111         @Override
112         public void actionPerformed(ActionEvent e) {
113            count = 0;
114            tfCount.setText(count + "");
115         }
116      }
117
118      /** The entry main() method */
119      public static void main(String[] args) {
120         // Run GUI codes in the Event-Dispatching thread for thread safety
121         javax.swing.SwingUtilities.invokeLater(new Runnable() {
122            public void run() {
123               new TestAction();  // Let the constructor does the job
124            }
125         });
126      }
127   }
```

4.4 WindowEvent & WindowListener/WindowAdapter

Interface `WindowListener` is used for handling `WindowEvent` triggered via the three special buttons (minimize, maximize/restore down, and close) on the top-right corner of the window or other means. There are 7 abstract methods declared in the interface, as

follows:

```
public interface WindowListener extends java.util.EventListener {
   public void windowClosing(WindowEvent evt);
      // called-back when the user attempt to close this window, most commonly-used handler
   public void windowActivated(WindowEvent evt);
      // called-back when this window is set to the active window
   public void windowDeactivated(WindowEvent evt);
      // called-back when this window is no longer the active window
   public void windowOpened(WindowEvent evt);
      // called-back when this window is first made to be visible
   public void windowClosed(WindowEvent evt);
      // called-back when the window has been closed
   public void windowIconified(WindowEvent evt);
      // called-back when this window is minimized
   public void windowDeiconified(WindowEvent evt);
      // called-back when this window is change from minimize to normal state
}
```

The most commonly-used method is `WindowClosing()`, which is called when the user attempts to close this window via the "window-close" button or "file-exit" menu item.

```
// Sample handler for windowClosing()
@Override
public void WindowClosing(WindowEvent evt) {
   // Ask user to confirm
   ......
   // Perform clean up operations
   ......
   System.exit(0);   // Terminate the program
}
```

`WindowAdapter`

A `WindowEvent` listener must implement the `WindowListener` interface and provides implementation to ALL the 7 abstract methods declared. An empty-body implementation is required even if you are not using that particular handler. To improve productivity, an *adapter* class called `WindowAdapter` is provided, which implements `WindowListener` interface and provides default implementation to all the 7 abstract methods. You can then derive a subclass from `WindowAdapter` and override only methods of interest and leave the rest to their default implementation.

This example shows how to extend a `WindowAdapter` class, using an anonymous inner class, to handle a window-closing event.

```
public class GUIApplication extends JFrame {
   public GUIApplication() {  // Constructor
      ......
      this.addWindowListener(new WindowAdapter() {
         @Override
         public void windowClosing(WindowEvent evt) {
            System.exit(0);
         }
      });
   }
   ......
}
```

`JFrame's setDefaultCloseOperation()`

In Swing's `JFrame`, a special method called `setDefaultCloseOperation()` is provided to handle clicking of the "window-close" button. For example, to *exit* the program upon clicking the close-window button, you can use the following instead of `WindowListener` or `WindowAdapter`.

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  // exit program upon clicking window-close button
```

Similarly, most of the event listener interface has its equivalent adapter, e.g., `MouseAdapter` for `MouseListener` interface, `KeyAdapter` for `KeyListener` interface, `MouseMotionAdapter` for `MouseMotionListener` interface. There is no `ActionAdapter` for `ActionListener`, because there is only one abstract method inside the `ActionListener` interface, with no need for an adapter.

A word of caution: If you implement the `WindowListener` yourself and misspell a method name says `windowClosing()` to `winowClosing()`, the compiler will signal an error to notify you that `windowClosing()` method was not implemented. However, if you extend from `WindowAdapter` class and misspell a method name, the compiler treats the misspell method as a new method in the subclass, and uses the default implementation provided by `WindowAdapter` for handling that event. This small typo error took me a few agonizing hours to debug. This problem is resolved via the annotation `@Override` introduced in JDK 1.5, which tells the compiler to issue an error if the annotated method does not override its superclass.

4.5 KeyEvent & KeyListener/KeyAdapter

The `KeyListener` interface defines three abstract methods:

```
void keyTyped(KeyEvent evt)
   // Called-back when a key has been typed (pressed followed by released)
void keyPressed(KeyEvent evt)
   // Called-back when a key has been pressed
void keyReleased(KeyEvent evt)
   // Callback when a key has been released
```

There are two kinds of key events:

1. The typing of a valid *character*, e.g., `'a'`, `'A'`. This is called a *key-typed* event.
2. The pressing or releasing of a key, e.g., up-arrow, enter, `'a'`, `shift+'a'`. This is a *key-pressed* or *key-released* event.

Use `keyTyped()` to process key-typed event, which produces a valid Unicode character. You can use `evt.getKeyChar()` to retrieve the `char` typed. `getKeyChar()` can differentiate between `'a'` and `'A'` (pressed shift+`'a'`).

You can use `keyPressed()` and `keyReleased()` for all the keys, character key or others (such as up-arrow and enter). You can use `evt.getKeyCode()` to retrieve the `int` Virtual Key (VK) code, e.g., `KeyEvent.VK_UP`, `KeyEvent.VK_ENTER`, `KeyEvent.VK_A`. You can also use `evt.getKeyChar()` to retrieve the unicode character, if the event produced a valid Unicode character.

`getKeyCode()` vs. `getKeyChar()`

- If you press `'a'` key, `getKeyChar()` returns `'a'` and `getKeyCode()` returns `VK_A`.
- If you press shift+`'a'`, two key-pressed events and one key-typed event triggered. `getKeyChar()` returns `'A'` and `getKeyCode()` returns `VK_SHIFT` in the first key-pressed event and `VK_A` in the second event. The first key-pressed event is often ignored by the program.

Notice that Virtual Key codes, key-char are used, instead of actual key code, to ensure platform and keyboard-layout independent.

For Example,

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /** Test KeyListener */
6  public class KeyListenerTest extends JFrame implements KeyListener {
7     /** Constructor to setup the GUI */
8     public KeyListenerTest() {
9        Container cp = getContentPane();
10       cp.addKeyListener(this);
11       // Need to enables receiving of key inputs for this GUI component.
12       cp.setFocusable(true);
13       // "May" need to request keyboard focus on this component.
14       cp.requestFocus();
15
16       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17       setTitle("Testing Key Listener");
18       setSize(300, 200);
19       setVisible(true);
20    }
21
22    @Override
23    public void keyTyped(KeyEvent e) {
24       char keyChar = e.getKeyChar();
25       System.out.println("keyTyped: Key char is " + keyChar);
26    }
27
28    @Override
29    public void keyPressed(KeyEvent e) {
30       int keyCode = e.getKeyCode();
31       char keyChar = e.getKeyChar();
32       System.out.println("keyPressed: VK Code is " + keyCode + ", Key char is " + keyChar);
33    }
34
35    @Override
36    public void keyReleased(KeyEvent e) {}  // Ignored
37
38    /** The entry main method */
```

```
39      public static void main(String[] args) {
40         // Run GUI codes on the Event-Dispatching thread for thread safety */
41         SwingUtilities.invokeLater(new Runnable() {
42            @Override
43            public void run() {
44               new KeyListenerTest();  // Let the constructor do the job
45            }
46         });
47      }
48   }
```

Try pressing 'a', 'A' (shift+'a'), enter, up-arrow, etc, and observe the key-char and VK-code produced by `keyTyped()` and `keyPressed()`.

Below is a sample handler for a key listener:

```
@Override
public void keyPressed(KeyEvent e) {
   switch (e.getKeyCode()) {
      case KeyEvent.VK_UP: ......; break
      case KeyEvent.VK_DOWN: ......; break
      case KeyEvent.VK_LEFT: ......; break
      case KeyEvent.VK_RIGHT: ......; break
   }
}

@Override
public void keyTyped(KeyEvent e) {
   // Can also be placed in keyPressed()
   switch (e.getKeyChar()) {
      case 'w': ......; break
      case 'a': ......; break
      case 'z': ......; break
      case 's': ......; break
   }
}
```

The commonly-used virtual key codes are:

- `VK_LEFT`, `VK_RIGHT`, `VK_UP`, `VK_DOWN`: arrow keys
- `VK_KP_LEFT`, `VK_KP_RIGHT`, `VK_KP_UP`, `VK_KP_DOWN`: arrow key on numeric keypad
- `VK_0` to `VK_9`, `VK_A` to `VK_Z`: numeric and alphebet keys. Also produce a Unicode char for the `getKeyChar()`
- `VK_ENTER`, `VK_TAB`, `VK_BACKSPACE`

4.6 MouseEvent & MouseListener/MouseAdapter

The `MouseListener` interface is associated with `MouseEvent` (triggered via mouse-button press, release and click (press followed by release)) on the source object. It declares 5 abstract methods:

```
public void mouseClicked(MouseEvent evt)
   // Called-back when a mouse button has been clicked (pressed followed by released) on the source
public void mouseEntered(MouseEvent evt)
   // Called-back when the mouse enters the source
public void mouseExited(MouseEvent evt)
   // Called-back when the mouse exits the source
public void mousePressed(MouseEvent evt)
   // Called-back when a mouse button has been pressed on the source
public void mouseReleased(MouseEvent evt)
   // Called-back when a mouse button has been released on the source
```

From the `MouseEvent` argument `evt`, you can:

- use `evt.getX()` and `evt.getY()` to retrieve the `(x, y)` coordinates of the location of the mouse.
- use `evt.getXOnScreen()` and `evt.getYOnScreen()` to retrieve the absolute`(x, y)` coordinates on the screen.
- use `evt.getClickCount()` to retrieve the number of clicks, e.g., 2 for double-click.
- use `evt.getButton()` to determine which button (`MouseEvent.BUTTON1`, `MouseEvent.BUTTON2`, `MouseEvent.BUTTON3`, or `MouseEvent.NOBUTTON`) is clicked.

An adapter class `MouseAdapter` is available, which provides default (empty) implementation to the 5 `abstract` methods declared in the `MouseListener` interface. You can create a mouse listener by subclassing the `MouseAdapter` and override the necessary methods. For example,

```
// Use an anonymous inner class (extends MouseAdapter) as mouse listener
aSource.addMouseListener(new MouseAdapter() {
   @Override
   public void mouseClicked(MouseEvent evt) {
      // ... do something ...
   }
});
```

4.7 MouseEvent & MouseMotionListener/MouseMotionAdapter

The `MouseEvent` is associated with two interfaces: `MouseListener` (for mouse clicked/pressed/released/entered/exited) described earlier; and the `MouseMotionListener` (for mouse moved and dragged). The `MouseMotionListener` interface declares 2 abstract methods:

```
public void mouseDragged(MouseEvent evt);
   // Called-back when a mouse button is pressed on a source and then dragged
public void mouseMoved(MouseEvent evt);
   // Called-back when the mouse has been moved onto a source but no buttons pressed
```

From the `MouseEvent` argument, you can use `getX()`, `getY()`, `getXOnScreen()`, `getYOnScreen()` to find the position of the mouse cursor, as described earlier.

The `mouseDragged()` can be used to draw a arbitrary-shape line using mouse-pointer. The mouse-dragged event starts when you pressed a mouse button, and will be delivered continuously until the mouse-button is released. For example,

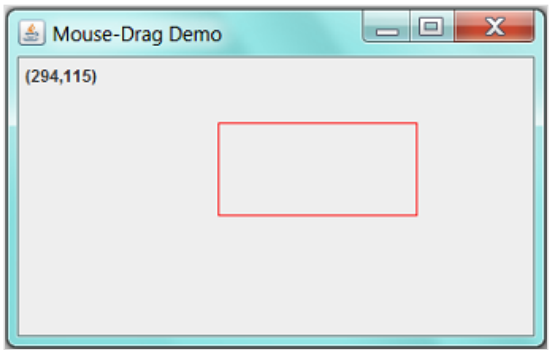
```
@Override
public void MouseDragged(MouseEvent evt) {
   // add evt.getX() and evt.getY() to a list of (x, y).
   // you can then use drawPolyline() method of the Graphics class to draw this line
   ......
}
```

Swing's `MouseInputListener/MouseInputAdapter`

Swing added a new event listener called `MouseInputListener` which combines `MouseListener` and `MouseMotionListener` as follows. You only need to implement one interface instead of two interfaces.

```
interface javax.swing.event.MouseInputListener
   extends java.awt.MouseListener, java.awt.MouseMotionListener {
   // Empty body - no additional method declared
}
```

Example: Using mouse-drag to draw a red rectangle. (You need to understand "Custom Graphics" - the next article - to read this program.)




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.event.*;
5
6  /** Test mouse-dragged */
7  @SuppressWarnings("serial")
8  public class MouseDragDemo extends JFrame {
9     private int startX, startY, endX, endY;  // of a rectangle
10    private JLabel statusBar;  // display the status
11
12    /** Constructor to setup the GUI */
13    public MouseDragDemo() {
14       // Define an anonymous inner class extends JPanel for custom drawing
15       // and allocate an instance
16       JPanel drawPanel = new JPanel() {
```

```
17          @Override
18          public void paintComponent(Graphics g) {
19             super.paintComponent(g);  // paint parent's background
20             g.setColor(Color.RED);
21             // drawRect() uses x, y, width and height instead of (x1,y1) and (x2,y2)
22             int x = (startX < endX) ? startX : endX;
23             int y = (startY < endY) ? startY : endY;
24             int width = endX - startX + 1;
25             if (width < 0) width = -width;
26             int height = endY - startY + 1;
27             if (height < 0) height = -height;
28             g.drawRect(x, y, width, height);
29          }
30       };
31
32       statusBar = new JLabel();
33       drawPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
34       drawPanel.add(statusBar);
35
36       // Allocate an instance of MyMouseDraggedListener
37       // and used it as MouseListener and MouseMotionListener
38       MyMouseDraggedListener listener = new MyMouseDraggedListener();
39       drawPanel.addMouseListener(listener);
40       drawPanel.addMouseMotionListener(listener);
41
42       setContentPane(drawPanel);
43       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44       setTitle("Mouse-Drag Demo");
45       setSize(400, 250);
46       setVisible(true);
47    }
48
49    private class MyMouseDraggedListener extends MouseInputAdapter {
50       @Override
51       public void mousePressed(MouseEvent evt) {
52          startX = evt.getX();
53          startY = evt.getY();
54          statusBar.setText("(" + startX + "," + startY + ")");
55       }
56       @Override
57       public void mouseDragged(MouseEvent evt) {
58          endX = evt.getX();
59          endY = evt.getY();
60          statusBar.setText("(" + endX + "," + endY + ")");
61          repaint();  // Called back paintComponent()
62       }
63       @Override
64          public void mouseReleased(MouseEvent evt) {
65          endX = evt.getX();
66          endY = evt.getY();
67          statusBar.setText("(" + endX + "," + endY + ")");
68          repaint();  // Called back paintComponent()
69       }
70    }
71
72    /** The entry main method */
73    public static void main(String[] args) {
74       // Run GUI codes on the Event-Dispatching thread for thread safety
75       SwingUtilities.invokeLater(new Runnable() {
76          @Override
77          public void run() {
78             new MouseDragDemo();  // Let the constructor do the job
79          }
80       });
81    }
82 }
```

[TODO] GUI programming is huge. Need to separate into a few articles.

REFERENCES & RESOURCES

1. "Creating a GUI With JFC/Swing" (aka "The Swing Tutorial") @ http://docs.oracle.com/javase/tutorial/uiswing/.
2. JFC Demo (under JDK demo "`jfc`" directory).
3. "SwingLabs" java.net project @ http://java.net/projects/swinglabs.
4. Java2D Tutorial @ http://docs.oracle.com/javase/tutorial/2d/index.html.
5. JOGL (Java Binding on OpenGL) @ http://java.net/projects/jogl/.
6. Java3D (@ http://java3d.java.net/).

Latest version tested: JDK 1.7
Last modified: April, 2012

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg) | HOME