Vertical Label 1.0 Component Specification

1. Design

Java provides a JLabel that can layout text in a horizontal manner but lacks the ability to layout text in a vertical manner. This component will provide functionality similar to a JLabel but will layout the text either in a 90 degrees or 270 degrees (i.e. where the ascent is on the left and the decent is on the right OR the ascent is on the right and the descent is on the left). A prototype screenshot is attached that shows the vertical label as it will be used in the UML Tool Side Menu.

The JVerticalLabel class extends the JLabel class, and it inherits all properties supported in JLabel (e.g. text, icon, alignment, font, color, component orientation etc.). It also provides a new flipped property to indicate the horizontal label should be rotated 90 degrees clockwise or anti-clockwise. The VerticalLabelUI is used by JVerticalLabel to paint the label.

1.1 Design Patterns

MVC Pattern - JVerticalLabel and VerticalLabelUI follows the MVC pattern used in Java Swing. The VerticalLabelUI is the view class, and the JVerticalLabel is the model class.

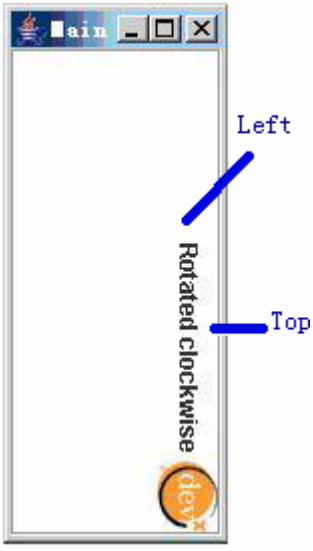
1.2 Industry Standards

Swing

1.3 Required Algorithms

1.3.1 Component Orientation, Horizontal and Vertical Alignments

Component orientation, horizontal and vertical alignments are applied to the vertical label as if the label is horizontal.



Picture 1

Picture 2

Example 1:

```
// Create a vertical label:

// The label's horizontal alignment value is RIGHT, its vertical
// alignment value is TOP, its component orientation value is
```

```
// RIGHT_TO_LEFT, and its flipped value is true indicating the label is
// rotated 90 degrees clockwise.

// The result should be like the label shown in Picture 1.
// From the picture it's quite clear that both horizontal and vertical
// alignments are relative to the label as if it's horizontal.
// And the icon is displayed on the right side of the label text as the
// component orientation value is RIGHT_TO_LEFT.

ImageIcon icon = new ImageIcon("icon.gif");
JVerticalLabel l = new JVerticalLabel("Rotated clockwise", icon,
           SwingConstants.RIGHT, true);
l.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
l.setVerticalAlignment(SwingConstants.TOP);
```

Example 2:

```
// Create a vertical label:

// The label's horizontal alignment value is RIGHT, its vertical
// alignment value is BOTTOM, its component orientation value is
// LEFT_TO_RIGHT, and its flipped value is true indicating the label is
// rotated 90 degrees clockwise.

// The result should be like the label shown in Picture 2.
// From the picture it's quite clear that both horizontal and vertical
// alignments are relative to the label as if it's horizontal.
// And the icon is displayed on the left side of the label text as the
// component orientation value is LEFT_TO_RIGHT.

ImageIcon icon = new ImageIcon("icon.gif");
JVerticalLabel l = new JVerticalLabel("Rotated clockwise", icon,
           SwingConstants.RIGHT, true);
l.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
l.setVerticalAlignment(SwingConstants.BOTTOM);
```

Picture 3

```
Example 3:
// Create a vertical label:

// The label's horizontal alignment value is LEFT, its vertical
// alignment value is TOP, its component orientation value is
// LEFT_TO_RIGHT, and its flipped value is false indicating the label is
// rotated 90 degrees anti-clockwise.

// The result should be like the label shown in Picture 3.
// From the picture it's quite clear that both horizontal and vertical
// alignments are relative to the label as if it's horizontal.
// And the icon is displayed on the left side of the label text as the
// component orientation value is LEFT_TO_RIGHT.

ImageIcon icon = new ImageIcon("icon.gif");
JVerticalLabel l = new JVerticalLabel("Rotated anti-clockwise", icon,
           SwingConstants.LEFT, false);
l.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
l.setVerticalAlignment(SwingConstants.TOP);



NOTE: According to the ComponentOrientation's class documentation, it is
used to encapsulate the language-sensitive orientation that is to be used
to order the elements of a component or of text. It's still valid for the
vertical label as the vertical label is simply a rotated view of the
horizontal label. The examples given above only show simple usage of this
property.
```

1.4 Component Class Overview

1.4.1 Package com.topcoder.gui.verticallabel

- **JVerticalLabel**: This class extends the JLabel class and it will lay out the text and icon of the label either in a 90 degrees or 270 degrees.
- **VerticalLabelUI**: This class extends the BasicLabelUI class and it's used to paint the icon and text of JVerticalLabel either in a 90 degrees or 270 degrees according to the JVerticalLabel's flipped value.

1.5 Component Exception Definitions

1.5.1 System Exceptions

- **IllegalArgumentException**: It is thrown when the passed-in argument is illegal. NOTE: A string is empty if its length is 0 after being trimmed.

1.5.2 Custom Exceptions

None.

1.6 Thread Safety

This component is not thread-safe. Both JVerticalLabel and VerticalLabelUI classes are mutable and not thread-safe.

Both JVerticalLabel and VerticalLabelUI classes are swing components, and they can be accessed by only one thread at a time in Java Swing Application, so they don't need to be thread-safe and VerticalLabelUI is ok to have mutable static variables too.

2. Environment Requirements

2.1 Environment

- Java1.5

2.2 TopCoder Software Components

None.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.gui.verticallabel

3.2 Configuration Parameters

None.

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

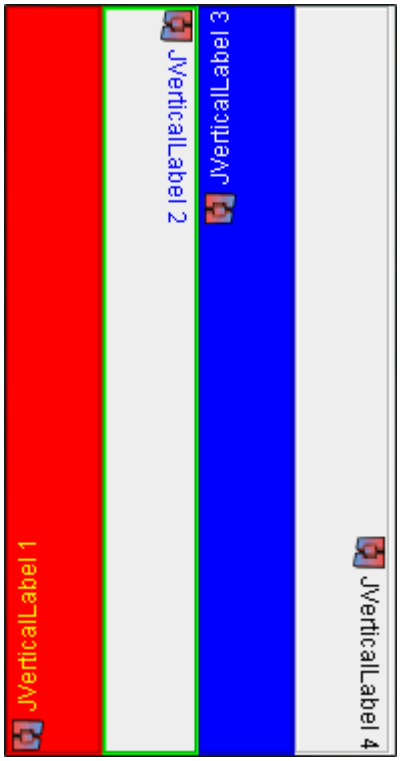
None.

4.3 Demo (updated)

```
ImageIcon icon = new ImageIcon("test_files/uml.png");

// Create the vertical labels with different horizontal alignment
// and flipped flag.
JVerticalLabel label1 = new JVerticalLabel("JVerticalLabel 1", icon,
      SwingConstants.LEFT, false);
JVerticalLabel label2 = new JVerticalLabel("JVerticalLabel 2", icon,
      SwingConstants.LEFT, true);
JVerticalLabel label3 = new JVerticalLabel("JVerticalLabel 3", icon,
      SwingConstants.RIGHT, false);
JVerticalLabel label4 = new JVerticalLabel("JVerticalLabel 4", icon,
      SwingConstants.RIGHT, true);

label1.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
label2.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
label3.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
label4.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);

label1.setVerticalAlignment(SwingConstants.TOP);
label2.setVerticalAlignment(SwingConstants.TOP);
label3.setVerticalAlignment(SwingConstants.TOP);
label4.setVerticalAlignment(SwingConstants.TOP);

// Set labels' background color.
label1.setBackground(Color.RED);
label2.setBackground(Color.GREEN);
```

```
label3.setBackground(Color.BLUE);
label4.setBackground(Color.WHITE);

// Set labels' foreground color.
label1.setForeground(Color.YELLOW);
label2.setForeground(Color.BLUE);
label3.setForeground(Color.WHITE);
label4.setForeground(Color.BLACK);

// Set labels' opaque.
label1.setOpaque(true);
label2.setOpaque(false);
label3.setOpaque(true);
label4.setOpaque(false);

// Set labels' font.
label1.setFont(Font.getFont("Arial"));
label2.setFont(Font.getFont("Courier New"));
label3.setFont(Font.getFont("Tahoma"));
label4.setFont(Font.getFont("Times New Roman"));

// Set label’s border.
label1.setBorder(new EtchedBorder());
label2.setBorder(new EtchedBorder());
label3.setBorder(new EtchedBorder());
label4.setBorder(new EtchedBorder());

// Show the vertical labels on UI.
Container c = new Container();
c.setLayout(new GridLayout(1, 4));
c.add(label1);
c.add(label2);
c.add(label3);
c.add(label4);

JFrame jframe = new JFrame("Demo");
jframe.setSize(new Dimension(200, 400));
jframe.setLocationByPlatform(true);
jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jframe.setResizable(false);
jframe.setContentPane(c);
jframe.setVisible(true);
```

The displayed label should be like:



5 Future Enhancements

None.