6.831 User Interface Design & Implementation
Fall 2004

PS4: Rotatable Labels

This assignment explores the following topics:

- delegation to other components (JLabel);
- bounding box computation;
- using coordinate transformations in drawing (rotation);
- direct-manipulation rotation.

In this problem set, you will implement a variant of JLabel that can display text rotated at an arbitrary angle. You will also implement a controller that allows the user to rotate a label with the mouse.

This problem set lays the groundwork for PS5, in which you will create automatic layout managers that adjust not only size and position but also rotation angle.

Provided Resources

We provide one class for this assignment:

- RLabel: skeleton for the rotating label.

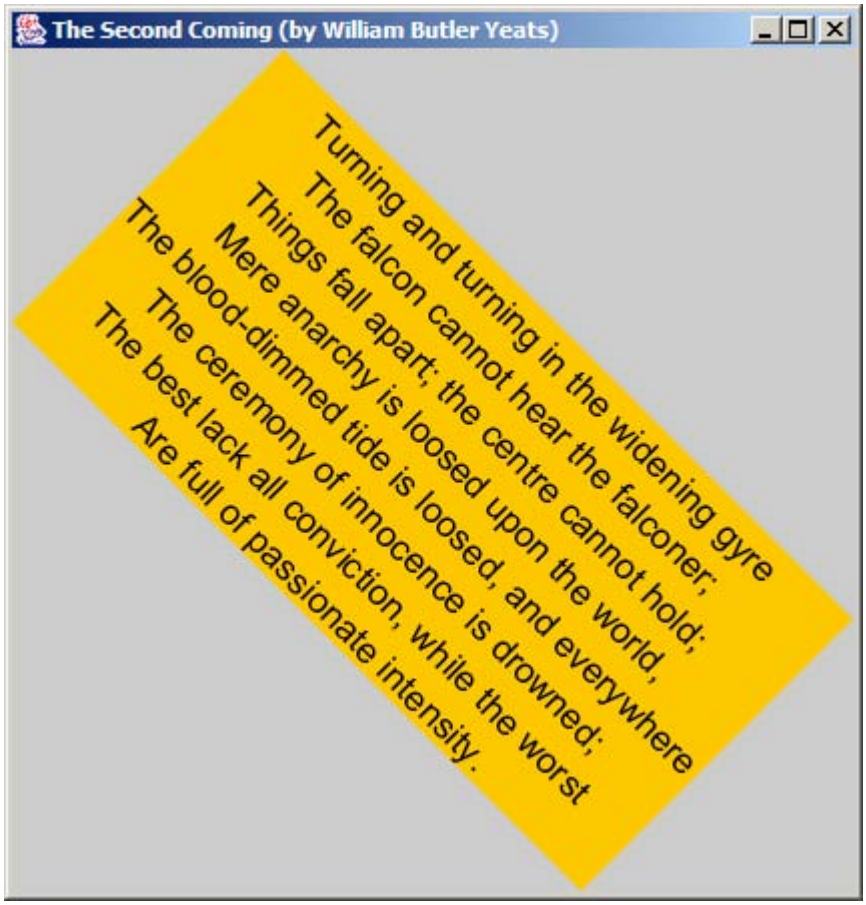
Problem 1: RLabel (40%)

The RLabel implementation we gave you ignores its angle property, instead always displaying the label with no rotation. Complete this class so that it draws itself at the appropriate angle.

RLabel should compute the correct bounding box for its own layout. It should draw the label centered in the bounding box, rotated by the given angle around that center. The comments in RLabel explain how the angle should be interpreted. RLabel's get and set methods should behave correctly, including updating the label appropriately when the text or angle are changed by setText() or setAngle().

You may find the Math class useful for trigonometric functions, Graphics2D essential for drawing with rotation and translation, and AffineTransform useful for reproducing the coordinate transformations done by Graphics2D.

RLabel has a main method for testing. Once you have finished making your changes, this main method should produce the following window:



Note that the text is surrounded by an orange rectangle. We did this just for debugging, so that it's easy to check that your RLabel is producing the correct bounding box. When the test window first appears, it should fit tightly around the RLabel (give or take a pixel or two, because of roundoff error).

RLabel.main() takes the angle of the label as an optional command-line argument (but be warned that this angle is measured in degrees, not radians). Test your RLabel at different angles to ensure that you're computing the correct bounding box and painting correctly.

Problem 2: RotateInteractor (30%)

Design and implement a RotateInteractor that can freely rotate RLabels. Your RotateInteractor should be able to listen to any number of RLabels, but since the user has only one mouse, RotateInteractor only needs to actively turn one label at a time. When the mouse is pressed and dragged on an RLabel (anywhere in its bounding box, not just on the orange rectangle), RotateInteractor should rotate it, following the mouse, until the mouse button is released. To determine how far the label should be turned, compute the angle between two lines: one from the center of the label to the point where the mouse was pressed, and the other from the center to the mouse's current position. You'll find Math.atan2() helpful for computing this angle.

Change RLabel's main method so that the test window uses a RotateInteractor to allow the label to be rotated freely. (Note: don't try to resize the window automatically to keep the RLabel tightly packed in it while it's being rotated. The intial window size should be tightly packed, but from then on, let the user control the window size.)

Questions (30%)

Answer the following questions in readme.txt.

1. In the code you've written, the JLabel is used merely as a delegate for painting -- i.e., as a stencil. Suppose instead that the JLabel is added as a *child component* of the RLabel. What would the RLabel have to do to ensure that the JLabel is correctly painted at the correct rotation? Think about both the initial paint and subsequent repaints.
2. Suppose we wanted to build an RScrollBar that could display a working JScrollBar at any angle of rotation. What would the RScrollBar have to do to make sure that the scrollbar's input handling worked properly?
3. Rotate your RLabel very slowly, a pixel at a time, and observe that the text doesn't turn smoothly. What happens? What does this tell you about how Swing draws text?

What to Hand In

Package your completed assignment as a jar file, as described in PS0. Here's a checklist of things you should confirm before you hand in:

- ☐ all your Java source is included in your jar file (Javadoc documentation isn't necessary)
- ☐ the main class of your jar file is RLabel
- ☐ all necessary third-party libraries are included, either inside your jar or as separate jars referenced by your jar's classpath
- ☐ any resources used by your code are included in the jar and referenced as resources
- ☐ readme.txt is included, and it answers the questions above and credits anybody you discussed the assignment with

Before you submit your solution, put all the jar files you plan to submit in an empty directory and make sure you can run it:

```
java -jar yourfile.jar
```