6.831 User Interface Design & Implementation
Fall 2004

PS5: Rotation-Aware Layout

This assignment explores the following topics:

- implementing automatic layout protocols;
- position, size, and orientation computation;
- declarative vs. procedural specification

In this problem set, you will implement two novel layout managers for rotatable RLabels. One layout manager arranges labels in a circle. The other layout manager arranges labels horizontally, tilting them if necessary to fit.

You have **two weeks** to complete this problem set.

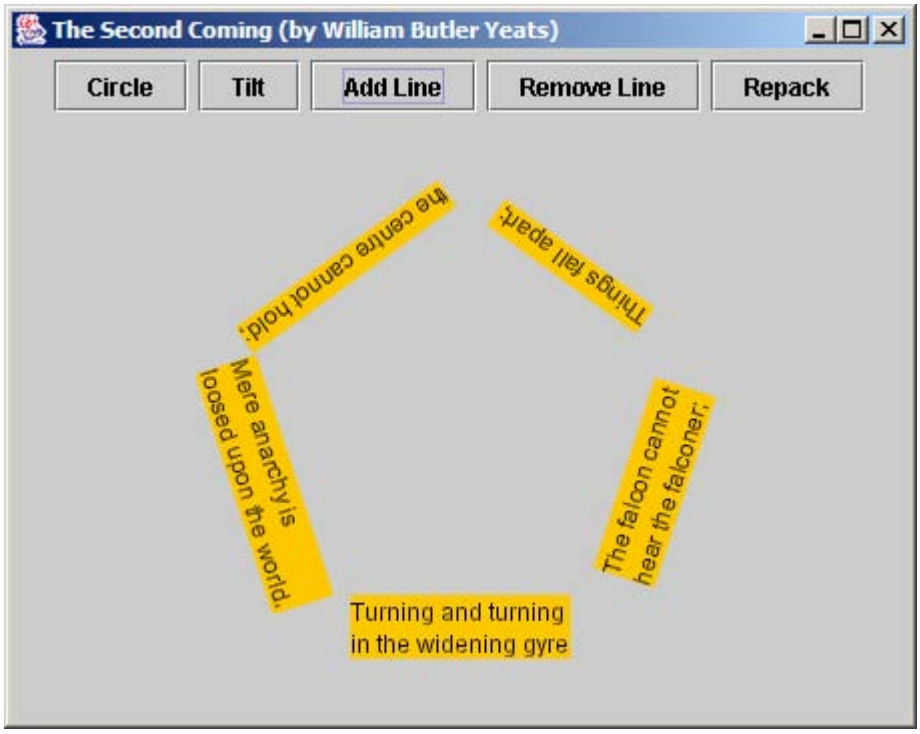
Provided Resources

We provide the following classes for this assignment:

- Circle Layout : skeleton for the circle layout manager
- TiltLayout : skeleton for the tilting layout manager
- LayoutTester: a class for testing CircleLayout and TiltLayout
- RLabel: skeleton for the rotating label.

Problem 1: CircleLayout (35%)

When CircleLayout is used as the layout manager for a container of RLabels, it arranges the labels in a circle, oriented to be read from outside the circle. This kind of layout would be useful for a GUI designed to be projected on a round tabletop with multiple users sitting around it. For example:



More precisely, CircleLayout should place the *n* labels in the container on the sides of a regular polygon with *n* sides. The polygon should be centered in the container, and its side length should be the width of the widest label. The top edge of each label's text should be centered on a side of the polygon. The first label in the container should be horizontal, on the bottom of the polygon, and successive labels should be positioned counterclockwise around the polygon.

When n = 2, the "polygon" is simply a horizontal line segment centered in the window, with the first label drawn below it and the second label drawn upside down above it. When n = 1, the single label is simply centered in the container.

For the purpose of this problem set, CircleLayout can ignore the actual space available in the container. If the container is too small, then the labels will simply be cropped.

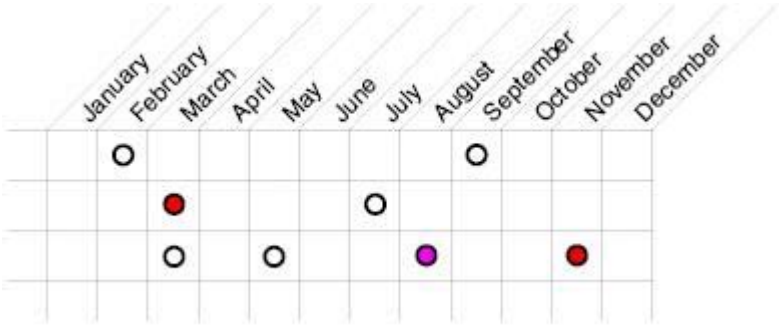
CircleLayout's behavior is undefined if the container has anything but RLabels in it. Don't worry about implementing preferred size or minimum size for CircleLayout.

Hints:

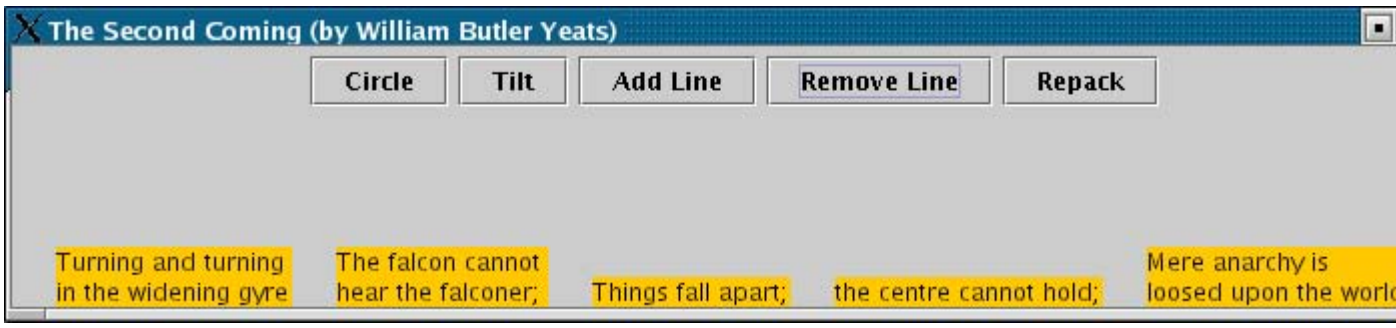
- Your layoutContainer() method must set three properties of each label: its position, its size, and its orientation.
- Remember that there's a difference between the size of the text (the orange rectangles in the figures above) and the size of the RLabel (returned by getSize() or getBounds()). Your code may have to make each RLabel temporarily horizontal in order to find out the true size of the text. If you find that distasteful, you may want to add another method to RLabel that gives the text size without forcing you to mutate the label.
- It might help to think about the circle circumscribed around the regular polygon.

Problem 2: TiltLayout (35%)

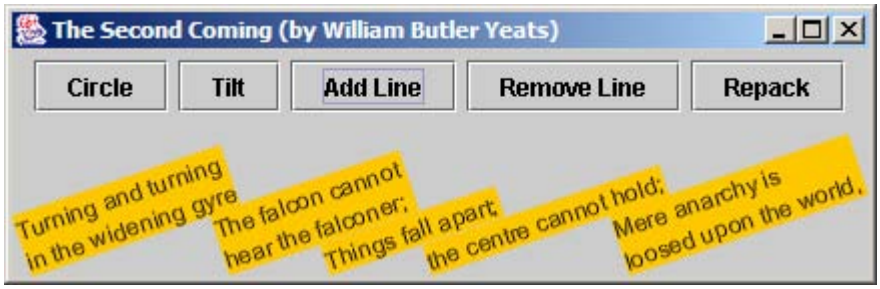
One motivation for TiltLayout is positioning the column headers of a table so that they don't affect the width of the columns. By tilting the column headers, we can push data items more closely together:



TiltLayout arranges RLabels in a row. As long as the labels fit horizontally, it displays them horizontally at their preferred size, flush against the bottom of the container. Any excess horizontal space is divided evenly among the gaps (left margin, right margin, and spaces between labels):



When the labels exceed the available horizontal space, however, the labels are tilted upwards and packed tightly into the space:



All labels are tilted at the same angle. The skeleton implementation of TiltLayout includes a method getTiltAngle() that computes a suitable approximation for the tilt angle.

As the available space shrinks, the tilt angle increases until the labels become completely vertical, at which point the container starts cropping from the right:

Don't worry about implementing preferred size or minimum size for TiltLayout.

Questions (30%)

Answer the following questions in readme.txt.

1. Propose two other ways that CircleLayout might behave when the container isn't large enough to display its preferred layout, and discuss their advantages and disadvantages.
2. TiltLayout makes an implicit assumption about the shape of the components it's arranging -- an assumption that makes sense for most RLabels but wouldn't necessarily be true for arbitrary components. What is it?
3. The provided implementation of getTiltAngle() makes another assumption about the RLabels it is laying out. Demonstrate a case where getTiltAngle() computes the wrong tilt angle.
4. Write a declarative specification for TiltLayout -- i.e., a set of constraint equations that relate the *x* coordinates, tilt angle, widths, and heights of the components. Don't worry about the horizontal or vertical cases; just the tilted case.

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 LayoutTester

☐ 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
```