CS 416 Program 1P: Wheels-like AWT

January 30, 2017

Important Dates and Requirements

- The program is due Monday, February 6, at 23:59. Late: -3 (Tu), -7 (We), -12 (Th), -20 (Fr)
- The last submission is Friday, February 10 at 23:59

1. Overview

This assignment is intended to give you practice using the basic features of AWT, especially its *Graphics2D* class. You will extend the set of classes that we built from the *Smart* classes described in the text and used in Lab 1, *AEllipse*, *ARectangle* and *ALine*. In this assignment you will extend *ARoundRectangle* from Lab 1 and create a new class, *ARegularPoly* that defines a *regular polygon.* Google "regular polygon" for explanations. The images at http://mathworld.wolfram.com/RegularPolygon.html match the orientation of my solution, but other orientations are acceptable as long as your objects are still regular polygons. You must also define two classes of *composite* objects containing multiple *A*-objects (such as the *AWTBot* of Lab 1). Both composite objects should have at least two instances of different *ARoundRectangle* objects. One composite object must have at least two instances of *ARegularPoly* objects with different numbers of sides.

2. The Program

1. Extend the implementation of the *ARoundRectangle* class from Lab 1 by adding the following methods:
   ```
   public ARoundRectangle( int x, int y, int w, int h, int arcW, int arcH );
   public void setArcSize( double arcWidth, double arcHeight )
   ```
   The two "arc" parameters are the width and height of the corner arcs. See the documentation for *java.awt.geom.RoundRectangle2D* for details about what these parameters mean.
2. Create a scene in *AWTPanel.java* composed of instances of the *AEllipse*, *ARectangle*, *ARoundRectangle,* and *ALine* classes. Use an *ArrayList<AShape>* or a *Vector<AShape>* to hold a list of references to *A*-objects that need to be displayed in the *paintComponent* method of *AWTPanel.*
3. Define a class, *First*, that *implements* the *AShape* interface and defines a composite graphics object containing at least one each of *AEllipse*, *ARectangle*, and *ALine*, along with at least 2 instances of *ARoundRectangle* with different sizes and arc parameters. The interface should match that of the *AWTBot* class in *Lab 1*. Add **at least 2 instances** of this object to your *AWTPanel* scene.
4. Complete a wheels-like class that models a *regular polygon* object. This class should be called *ARegularPoly* and *implement* the *AShape* interface. It should use an instance of *java.awt.Polygon* as the mechanism for drawing its regular polygon. *ARegularPoly* should allow the application to define arbitrary regular polygons by specifying a location, the number of sides and the radius (the distance from the center to each point of the polygon. The starting version of *ARegularPoly* has 1 constructor and 16 other *public* methods, most of which are identical or nearly identical to code in *ARectangle* and *AEllipse*. 11 of these methods are very short *get* or *set* methods, most of which are similar (or identical) to the corresponding methods in *AEllipse* and *ARectangle* and are complete. You must implement or complete:
   ```
   private void makePolygon( int x, int y ) // create java.awt.Polygon
   public void fill( java.awt.Graphics2D brush2 )
   public void draw( java.awt.Graphics2D brush2 )
   public void setSize( int w, int h )
   public void setRadius( int radius )
   public void setRotation( int deg ) // rotate poly to orientation deg
   ```
   You will have to read the Java documentation for *java.awt.Polygon* and the comments in the starter code.
5. Create a second composite object class, called *Second* that *implements AShape* and contains at least 2 instances of different (side count) *ARegularPoly* objects and at least 2 different *ARoundRectangle* objects.
6. The easiest implementation of a regular polygon is to generate *double* arrays holding the vertices of the regular polygon whose **center** is at (0,0). This is already done in the starter code by the method *makeVertices*, which also computes the upper left corner of the bounding box of this polygon, which is the location of the polygon.
7. You should not change the implementation of *AWTApp, AEllipse, ARectangle,* or *ALine* and you should not change the *public* interface defined in the starter code for *ARoundRectangle* and *ARegularPoly*.

1P 1P 1P 1P 1P

3. Testing strategy

The major testing strategy for this assignment is to use *unit tests* and *driver test* programs:

1. Unit tests: implement *First.main* and *Second.main* methods that at least use every method of the *First* and *Second* classes, respectively.
2. Expand the skeleton test program *RoundTest.java* to create lots of instances of *ARoundRectangle* that demonstrate the effects of different parameter values for the *setArcSize* method, different locations, sizes, colors, etc. Your resulting image should convince the viewer (i.e., the grader) that you have thoroughly tested your class.
3. Complete the *RegularPolyTest* class. The code you add should thoroughly test your *ARegularPoly* code with different parameters. **At the minimum it should guarantee that every *public* method of your class is invoked at least once – including those you did not change;** you have to verify that your code does not break the existing code.

4. Notes

1. Use the online Java API documentation to learn about the features of *java.awt.Polygon*.
2. You can download the starter code from *~cs416/public/1P/*.
3. Note that there are inconsistencies in the *awt* classes we are using as foundations for the *A*-classes:
   - *Rectangle2D.Double* and *Ellipse2D.Double* are in the *java.awt.geom* package, whereas the *Polygon* class is in the *java.awt* package;
   - *Rectangle2D* and *Ellipse2D* use floating point numbers for coordinates; *Polygon* uses *int*;
   - *Rectangle2D* and *Ellipse2D* have a location that can be changed; *Polygon* does not;
   - *Rectangle2D* and *Ellipse2D* have a size that can be changed; *Polygon* does not.

   The *ARegularPoly* specification supports the notion of a "location" of the polygon and allows the orientation of the polygon to be changed via the *rotation* method and the size to be changed via the *setSize* method. You have to implement these behaviors. *setSize* is a bit of a problem; see comments in starter code.
4. The *java.awt.Polygon* method *getBounds* returns a *java.awt.Rectangle*. This object defines the smallest axis-aligned rectangle that encloses the vertices of the polygon. We'll interpret the upper left corner as the "location" of the polygon. The *java.awt.Rectangle* class has methods, *getX(), getY(), getWidth(),* and *getHeight()* that return *double* values.
5. Whenever you change the vertices of the polygon (as happens in *setRotation* and *setSize*), you have to invoke the `Polygon.getBounds()` method so the *Polygon* object will recompute its bounds.
6. The *setRotation* and *setRadius* methods require you to recompute all the vertices of the polygon and either update the existing vertices or generate a new *java.awt.Polygon* object. Read the comments in the starter code for these methods for more help.
7. Make sure your code is well-modularized. In other words, keep methods short and create appropriate *private* methods so that code is not replicated.
8. **Pay attention to the style conventions. We'll use the *checkstyle* program to grade for style.**

5. Point allocation ("multiple" implies using different parameters for the different instances)

15 *AWTPanel* scene: multiple round rects, reg polys, other objects
10 *First* scene: multiple round rects, other objects
10 *Second* scene: multiple round rects, reg polys, other objects
15 *RoundTest* scene: show thorough testing of *ARoundRectangle* methods/parameters
35 *RegularPolyTest* completion that shows a thorough test of your *ARegularPoly* class
- 20 basic methods/parameters (excluding *setRotation* and *setSize* methods)
- 15 *setRotation* and *setSize* for *ARegularPoly* implemented and thoroughly tested.

The correctness of your code will be tested by independent tests we will provide; points will be deducted from the points awarded in the categories listed above that might only test whether you implemented the features.

Submission

Submit your assignment as 1P. Do **not** submit *AShape, AEllipse, ARectangle, ALine,* or *AWTApp*.