Circular Tracks. Part 1

Due Date

**Monday, November 16, 2020**

11:59 PM

This is another “relaxation machine”, which should have a calming effect if you stare at it for a while. For this program, you should first thoroughly understand the code for the TracksAnimation program. In this new version, the frame should be 600 pixels x 600 pixels. The class CircularTracksShape will implement Moveable to create and manipulate the “shape” icon that the class ShapeIcon paints.

It should be straightforward to have the draw() method in the class CircularTracksShape lay down the multicolored circular tracks: you just need a loop to paint partially overlapping circular disks centered at (300,300); the diameters of consecutive disks differ by 20 pixels. Although the photo doesn’t show this, you should also color in the 4 “corner” regions beyond the outer track: you can choose the color scheme.

How do we draw the little trains that circulate on the tracks? Consider the track T\_i whose inner radius is 20\*I pixels and whose outer radius is 20\*(i+1) pixels. The center of the circular train car traveling on this track, call it C\_i = (a\_i, b\_i), is moving around a circle centered at C(300,300) with radius r\_i = 20\*I + 10 pixels. Consider the instant when the radius drawn from C to C\_i makes an angle theta[i] with the horizontal line running through C in the positive-x direction (to the right). Now imagine the little circular car centered at C\_i which fits snugly into T\_i as it travels around this track. By extending the radius until it touches the outer edge of the track, it is clear that the diameter of the car is 20 pixels. To paint this car in Java, we need to use the usual approach: what are the coordinates of the upper left-hand corner of the square which encloses the car (so that the car is inscribed: it touches each side of the square once)? Since the car is inscribed in the square, and since the diameter of the car is 20 pixels, the upper left-hand corner of the inscribing square must be P\_i(a\_i – 10 , b\_i – 10) (remember that to move up the frame 10 pixels, you subtract 10 from the y-coordinate). Now, in the usual way, the coordinates of C\_i are a\_i = 300 + r\_i cos(theta[i]) and b\_i = 300 + r\_isin(theta[i]). From this, we can compute the coordinates of P\_i. Since the inscribing square has sides of length 20 pixels each, we can now draw the circular car. In the “draw” method, all of the above can be coded in a “for” loop with index i, one iteration for each track. Inside the loop, we can also pick a color for the car, either a constant color or one that keeps changing with each re-painting. For Part 1, you are NOT required to create the 2-car trains on the outer tracks. If you can create that feature on you own before Part 2 is announced, that would count for extra credit. Adding a second, or additiona,l cars, requires a little trigonometry-adjoining cars should touch at just one point.

How do we animate the cars? We want the outermost train to move counterclockwise and we want trains on adjacent tracks to move in opposite directions. As usual, we will use the “translate” method to move the cars around the tracks. As before, In the main program we will declare a timer, and in the timer-tick event handler we will call “repaint” and “translate”. Referring to angle theta[i] from above, if the car on this track is moving counterclockwise around its track, we want to modify theta[i] slightly each time “translate” is called. To create a “pleasing” animation (a matter of taste), we might have the outer cars moving more quickly than the inner cars-otherwise the inner cars would spin around very rapidly. So we might try

theta[i] = theta[i] + (something roughly proportional to i)\*pi/300.

You should experiment with the DELAY constant for the timer, with the proportionality factor, and with the angle pi/300 (arbitrary), to find a variable timing for the cars so that their movements are “smooth”. In particular, the innermost cars should move quite slowly, so that their movement can be clearly discerned. For trains moving in a clockwise direction, instead of adding to theta[i], subtract.

The trains have to be given an initial position, which can be determined by setting each theta[i] = 0 outside of the translate and draw methods. This way, the trains will start off on the horizontal axis, and will also periodically simultaneously return to their starting positions; in general, there is some nice regularity to the overall motion, but the amount of regularity depends on the proportionality factor- experiment. (Of course if you use theta[i] = theta [i] + c and theta[i] = theta[i] – c, where c is a constant, the motion is *too* regular- the cars are located along the same radial lines, not very interesting.)

--------------------------------------------------------------------------------------------------------------------------------------

1) Implement the animation program described in the word file and in the photo of the circular tracks. The Word file mentions some extra credit- ignore that, there is no extra credit component. Submit the following files with these exact names (no other names will be accepted):

- CircularTracksShape.java

- CircularTracksAnimationTester.java

2) A square tracks program is included for reference:  a photo, a pdf of the code, java code for the tracks icon (shape) and the testing program. By studying this code carefully, you will have a substantial hint for your program. Working with  circular tracks is actually considerably easier than working with square tracks.

\*This counts as 25% of exam 2. For Part 1, the trains just have 1 car each, NOT two cars.