Bootcamp Java Fall 2019
WarmUp for Project A

Analysis of Project

-This project called Brownian motion aims to replicate the brownian motion using Java programming commands.

INPUT (by User) :

1. Number of particles
2. Number of Steps per Walk

PROCESS:

1. Generate length of each step using the randomise operator(randomised between 5-15)
2. Generate angle of movement in degrees using randomise operator(randomised between 0-359)
3. Resultant color of path:
    - declare int red, int blue, int green
    - Randomise value of red, green and blue using the randomise operator(randomised between 0-255)
    - Generate resultant color using the constructor: new color(red, green, blue);
4. To determine next position:
    - Assuming (x0,y0) to be the starting point, the next point can be determined by using mathematical formulas uniquely determined for each quadrant(presented later in document), since angle input by the user can be any angle between 0-360 degrees.
5. Calculate the length of each walk by using the following formula:
    - Double total= totalsteps * length;
      Where ‘totalsteps’ is the number of steps by each particle and ‘length’ is the length of each step.
6. Calculate the Min and Max walk length using following steps:
    - There will be a bigger loop. The number of times the loop runs is the same as the number of particles entered by the user since that is also the number of paths to be generated.
    - Using the formula given in step 5, in each run of the loop, the length of each walk will be calculated
    - Simultaneously, in each run of the loop, the length of each of the walks can be taken as input to the subsequent elements of an array called ‘walklengths’.
    - After the end of the process of replicating the brownian motion, we can run a seperate set of commands to compare elements of the ‘walklengths’ array and find out the min and max walk length.

OUTPUT:

1. Minimum walk length
2. Maximum walk length
3. A variety of paths that resemble the brownian motion!

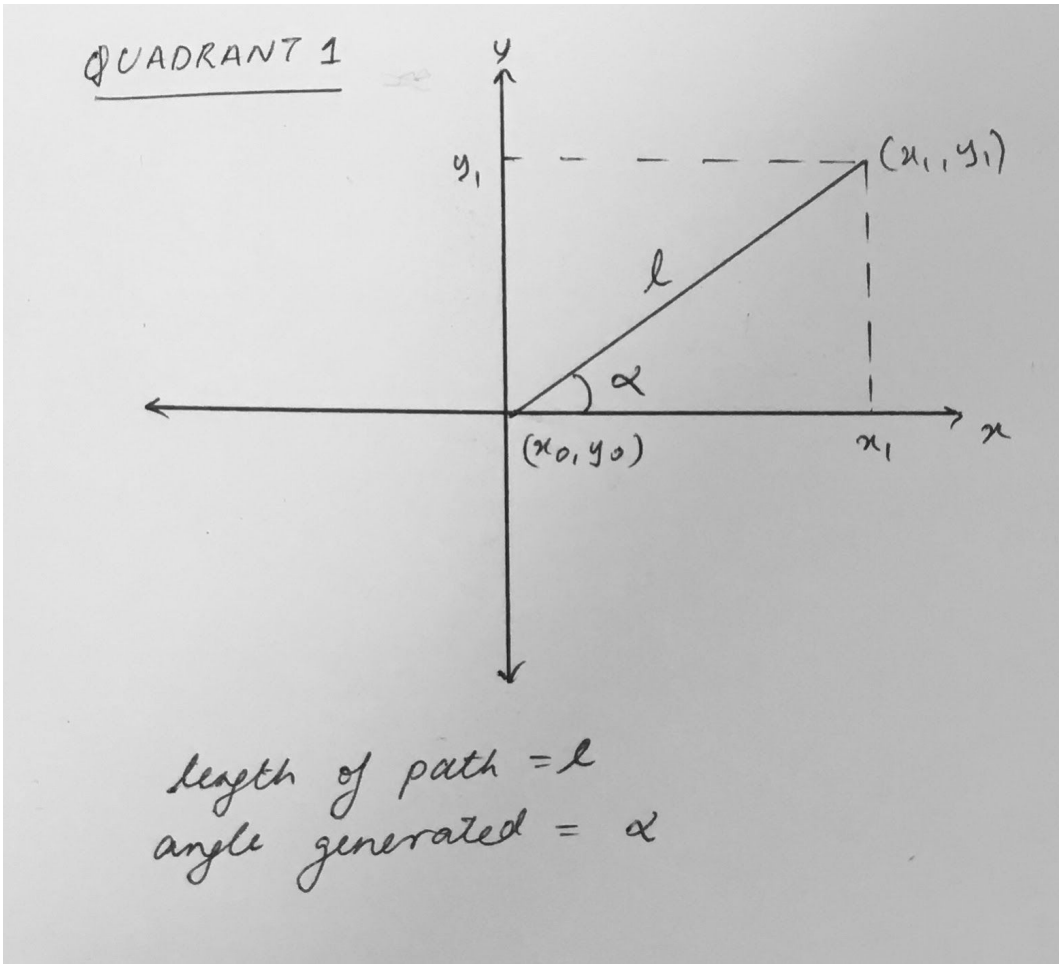
Geometric analysis on next page

GEOMETRIC ANALYSIS

Objective: To determine end point of walk, taking input of a randomly generated angle varying between 0-360 degrees and assuming initial position (xo,yo) to be the origin.

-In the actual code, we need to have import java.lang.math so as to include standard features of the math library, and specifically, to convert generated angle in degrees to radians as sin and cos functions take input parameters in radians.

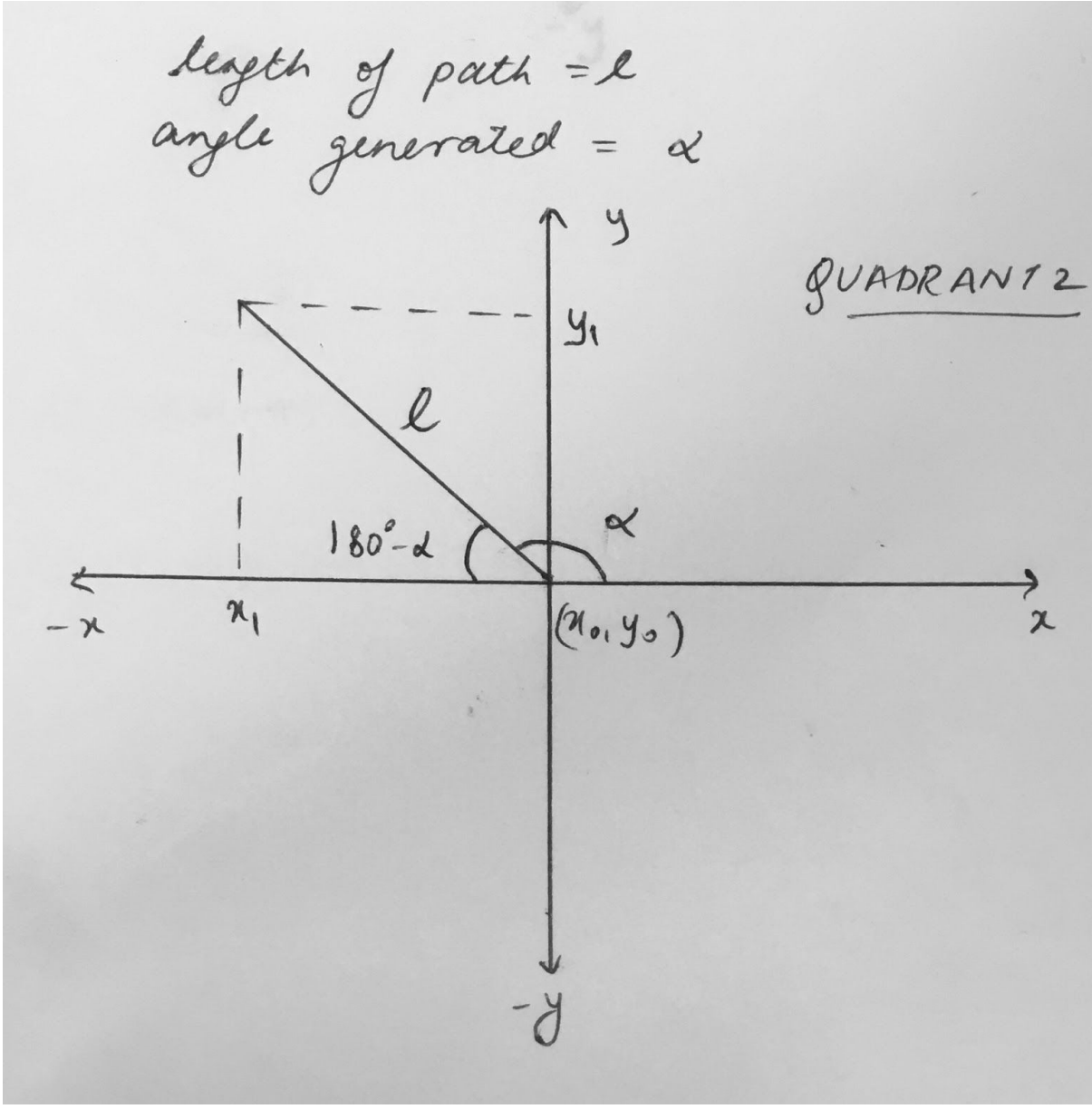
1. Quadrant 1: If angle generated is between 0-90 degrees



- sin(angle in degrees)= (y1-y0)/l
  So, y1=l*Math.sin(Math.toRadians(angle)) + y0; // all angles are converted to radians only in final line of code as shown.

- cos(angle in degrees)= (x1-x0)/l
  So, x1=l*Math.cos(Math.toRadians(angle)) + x0;

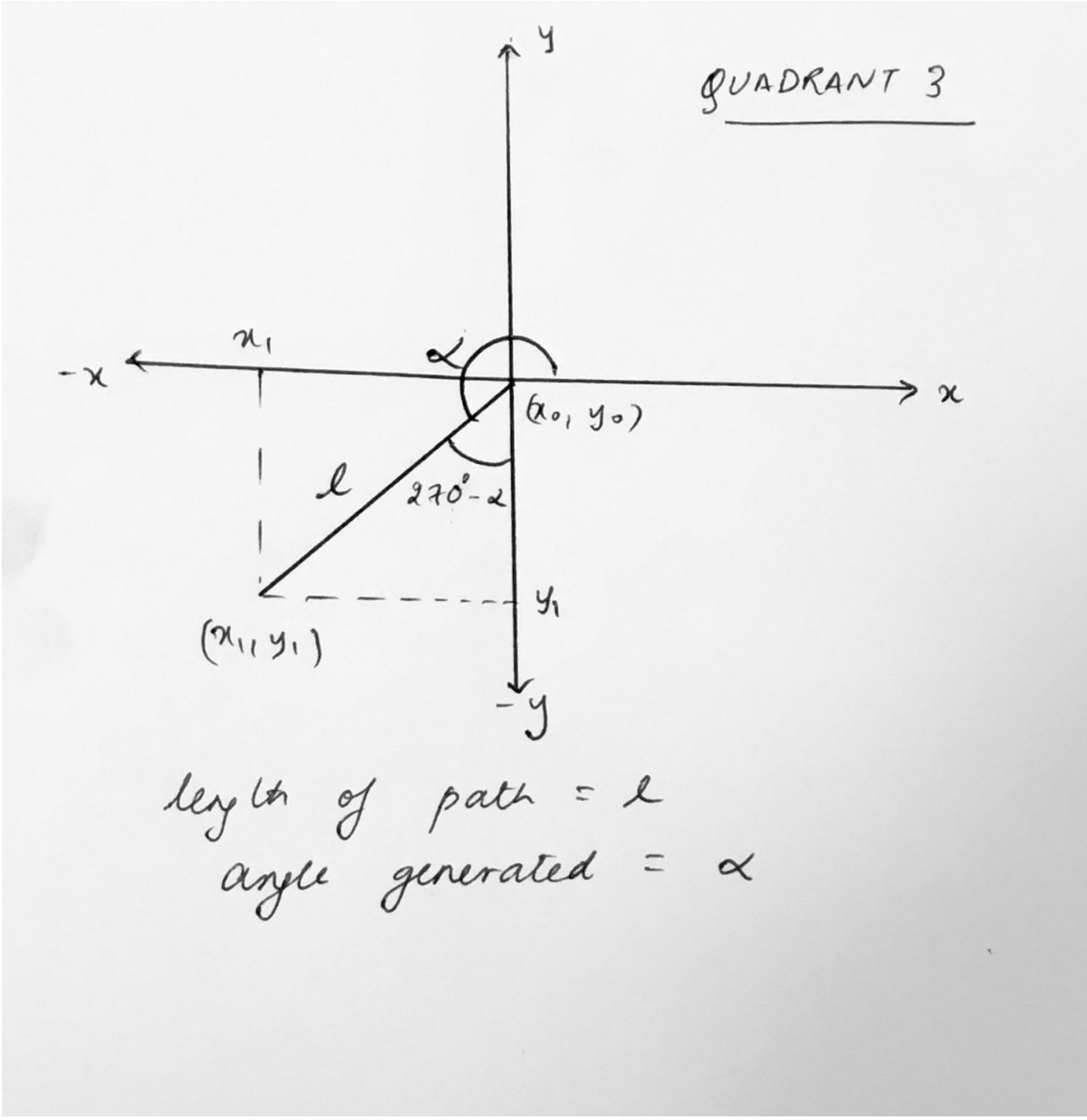
2. Quadrant 2: If angle generated is between 90-180 degrees



- sin(180 - angle in degrees)= (y1-y0)/l
  So, y1=l*Math.sin(Math.toRadians(180-angle)) + y0;

- cos(180 - angle in degrees)= (x0-x1)/l
  So, x1=x0 - l*Math.cos(Math.toRadians(180 - angle));

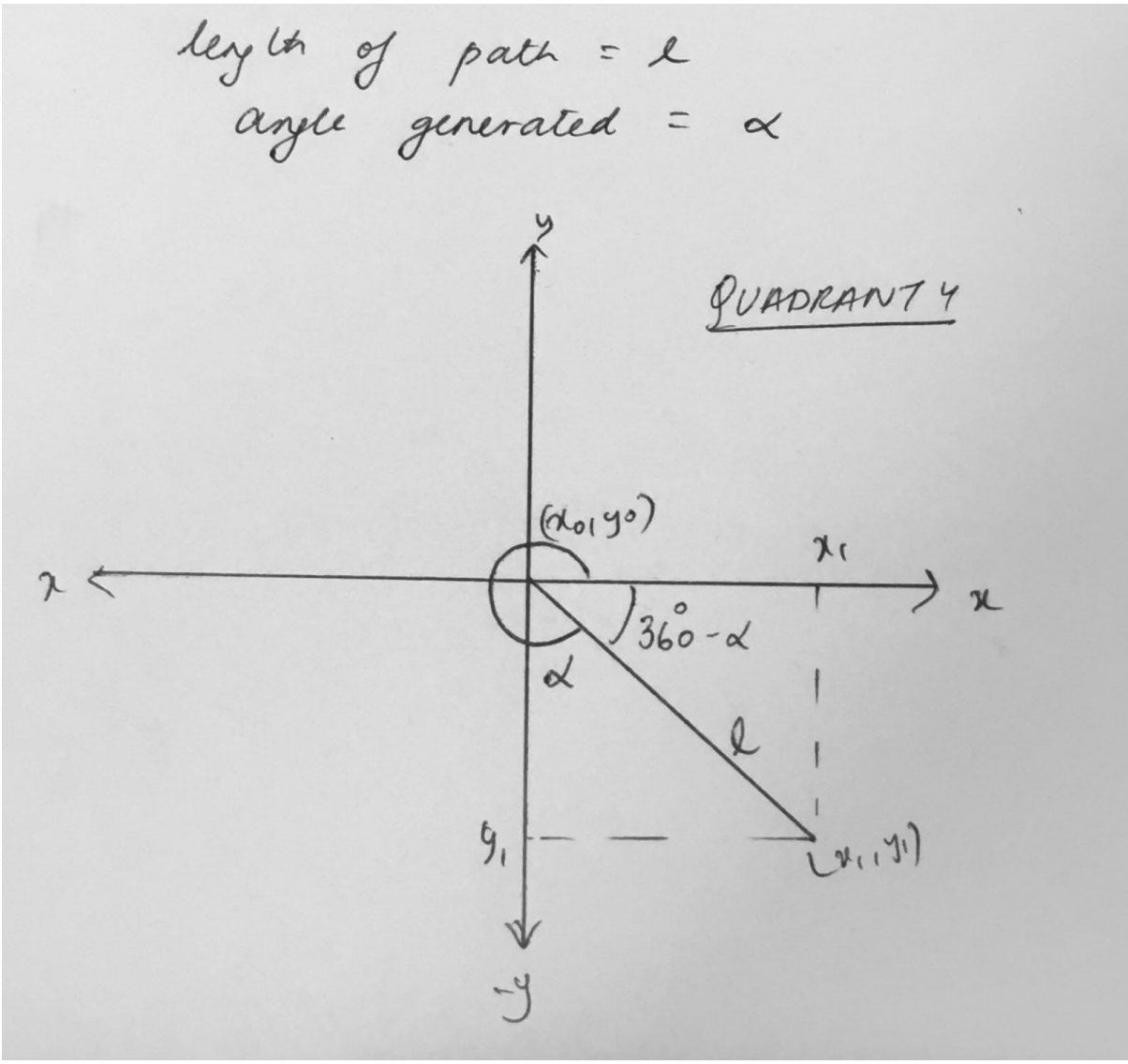
3. Quadrant 3: If angle generated is between 180-270 degrees



- cos(270 - angle in degrees)= (y0-y1)/l
  So, y1=y0 - l*Math.cos(Math.toRadians(270 - angle));

- sin(270 - angle in degrees)= (x0-x1)/l
  So, x1=x0 - l*Math.sin(Math.toRadians(270 - angle));

4. <u>Quadrant 4</u>: If angle generated is between 270-360 degrees



- sin(360 - angle in degrees)= (y0-y1)/l
  So, y1=y0 - l*Math.sin(Math.toRadians(360 - angle));

- cos(360 - angle in degrees)= (x1-x0)/l
  So, x1=l*Math.cos(Math.toRadians(360 - angle)) + x0;

End