Assignment 4
Etoy Programs
Due Dec 12

We are going to implement part of a program inspired by Etoys (http://www.squeakland.org), Scratch (https://scratch.mit.edu) and turtle graphics (http://en.wikipedia.org/wiki/Turtle_graphics).

We will have a turtle object that moves on the 2D plane. At the start of a program we can add labels points on the plane.

We will support the turtle moving and turning. The goal is to create a small language that kids can use to program the turtle. We will not implement the graphics part of the program, even though that is a key part of EToys, Scratch and Turtle graphics. We are just interested in the section of the program that interprets the language. The language is given below.

Mini Turtle Language

The turtle language consists of the following commands:

1) move X
   Move the turtle X units in the current direction. X is a positive integer. When the turtle moves it always ends up on integer coordinates. If when computing the move to the next location the result is not on integer coordinates round the x & y coordinate to the nearest integer value.

2) turn X
   Turn the direction of the turtle X degrees.

3) repeat k

```
repeat k
    statement1
    statement2
    ...
    statement
end
```

   k is an integer. The statements inside the repeat block are any legal statement in the language. the repeat statement has the obvious semantics.

4) #K = Y

   Y is an integer. The variable #K can be used in move, turn and repeat commands. All variables are one character long

5) distanceTo labelOfPoint

   Distance to the indicated point from the turtle's current position.

6) bearingTo labelOfPoint

   Angle to the indicated point from the turtle's current position.

6) #Pa = x, y

   Defines a point with label "a" with given x & y coordinates. x & y are integers.

When a program starts the turtle is in the middle of the screen headed horizontally right (zero degrees) and the pen is up. Here are two sample turtle programs which draw a square.

```
#side = 15
move #side
turn 90
move #side
turn 90
move #side
turn 90
move #side
```

Nearly the same program using the repeat statement.

```
#side= 10
repeat 4
    move #side
    turn 90
end
```

A program that defines two points and outlines a square around the first point

```
#Ps = 10, 10
#Pt = 10, 20
#d = distanceTo #s
#a = bearingTo #s
turn #a
move #d
#u = bearingTo #t
```

```
move 5
turn 90
move 5
repeat 4
    turn 90
    move 10
end
```

Turtle Class

The full program will have a Turtle class that draws on the screen. The Turtle class has fields to hold the current direction (in degrees) of the turtle and the current location (X & Y coordinates). The class has the following methods.

void move(int distance)
: Move the turtle distance units in the current direction and draw on the screen if the pen is down.

void turn(int degrees)
: Add “degrees” to the current heading of the turtle.

float direction()
: Returns the current direction of the turtle.

Point location()
: Returns the current location of the turtle.

float distanceTo(Point)
: Returns the distance to the given point

float bearingTo(Point)
: Returns the angle in degrees to the given point

Our goal is to execute a turtle program using Interpreter design pattern. The focus is not on the graphics part nor on the reading the turtle program rom file. So we will create a mock Turtle class that has the same fields and methods as the turtle class but will not draw on the screen. All the methods perform the same operation as the real Turtle class except for the move method. This will allow us to test our code without having to draw on the screen.

void move(int distance)
: Move the turtle distance units in the current direction and updates the field(s) indicating the current location of the turtle. The method does not draw on the screen instead it prints the current location of the turtle after the move.

Geometry & Computing New Location

The turtle starts at the location (0, 0), which if we were going to draw on the screen would be in the center of the screen. The turtle also starts with a direction of zero degrees. A direction of 0 degrees is to the right parallel to the X-axis. A direction of 90 degrees is straight up parallel to the Y-axis. A direction of -270 degrees is also the same as a direction of 90 degrees.

Now if the turtle is at (10,20), heading in a direction of 30 degrees and travels 15 units how do we compute where it ends up? First we have to convert the degrees to radians via the formula:

```
radians = degrees * pi / 180
```

Then we can get the deltaX and deltaY using:

```
deltaX = cosine(radians) * distance
deltaY = sine(radians) * distance
```

So in our example we get

```
x = 10
y = 20
degrees = 30
distance = 15
radians = pi*degrees/180 => 0.5236
deltaY = sin(radians) * distance => 7.5
deltaX = cos(radians) * distance => 12.9904
newX = x + deltaX => 22.9904
newY = y + deltaY => 27.5
So the turtle will end up at  (23, 28)  rounding  (22.99, 27.5)
```

Computing bearingTo

Computing the bearingTo a point is done using inverse (or arc) tan. If the turtle location is (Tx, Ty) and the points (Px, Py) then the bearing to the point is arcTan((Py - Ty)/(Px -Ty)). In Java tan returns radians so convert to degree Math.toDegrees(Math.atan((Py - Ty)/(Px -Ty))).

The Assignment

1. Using the interpreter pattern we want to take the following two turtle programs and represent the program as abstract syntax trees which we can execute. So if our file contained the following program after Reading (reading part is not necessary) and executing the program the turtle would end up at (23, 28).

```
move 10
turn 90
move 20
turn -60
move 15
```

the other program is the following square example:

```
#Ps = 10, 10
#Pt = 10, 20
#d = distanceTo #s
#a = bearingTo #s
turn #a
move #d
#u = bearingTo #t
move 5
turn 90
move 5
repeat 4
    turn 90
    move 10
end
```

Grading

Item	Points
Working Code	10
Unit Tests	10
Proper implementation of the Pattern	70
Quality of Code	10