CSc 110 Assignment 5
Slugs: Implementing File I/O and Arrays

Due:

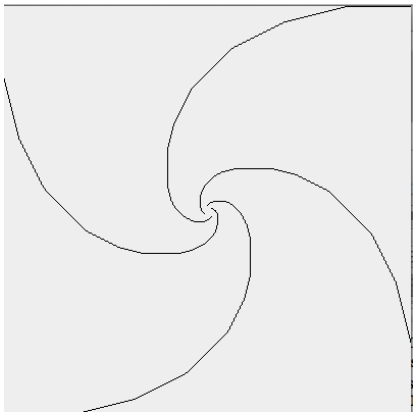
Friday Nov. 7, 8pm.

Learning Outcomes:

When you have completed this assignment, you should understand:

- Basic I/O processing of reading a text file and writing to a text file.
- The creation and manipulation of a simple array of Objects.
- The java Point object.
- The translation of basic graphs and trigonometry into computer graphics.
- Simple Program design.

Problem Description:



Sometimes computer programs are written to simulate the natural world. In this assignment, we simulate the movement of four slugs in a square sandbox. Initially positioned in separate corners of the sandbox, each slug travels at a constant speed in the direction of the slug that is located in front of it, in a counter-clockwise direction. The slugs keep moving until another move will result in the crossing of slug paths.

Computer Simulation

We can simulate constant movement by creating *time slices* of movement, so that for every single unit of time, each slug travels distance *d*.

The computer screen uses a coordinate system with an *x* and *y*-axis, and a grid of pixels, similar to a piece of graph paper. This coordinate system is equivalent to the system we learned in grade school, with one exception: the origin point (0,0) is located in the **top** left hand corner. So instead of pointing upwards, the *y*-axis points down. Any point on the coordinate system is represented as (x,y). The distance between any two points (x_1,y_1) and (x_2,y_2) is measured by the formula:

$$\sqrt{(x_2 - x_1)^2 + \langle y_2 - y_1 \rangle^2}$$

Once we are provided with the unit movement distance *d* (in pixels), and the dimensions of the sandbox (also in pixels), then we can program the simulation.

Representations

1. The sandbox is represented as a square of size *boxSize* that can fit on a computer screen. The upper left corner of the sandbox is represented by the point (0,0), the upper right (*boxSize*,0), the lower left (0,*boxSize*) and the lower right (*boxSize*,*boxSize*).
2. Each slug is represented by its current location, a point inside the sandbox. Java has a Point object that works perfectly as a slug position (see java.awt.Point).
3. The set of four slugs is represented by an array of Point objects.

The initial positions are as follows:

```
slug[0].x = 0;          slug[0].y = 0;
slug[1].x = 0;          slug[1].y = boxSize;
slug[2].x = boxSize;    slug[2].y = boxSize;
slug[3].x = boxSize;    slug[3].y = 0;
```

Implementation

During each iteration, each slug moves the unit movement distance *d* in a straight line from its current location to a new point, directly towards the next slug. (Note that if we make the distances small enough, the set of straight line segments looks like a curved line.) We keep track of the travel segments by printing the little lines as a set of four integers. For example the line:

```
0 200 1 175
```

represents start point (0,200), end point (1,175), and the line segment between the points. After each line is recorded, the value of slug[i] will be that of the last point.

Stopping:

The slugs stop moving when the distance from each slug[i] to slug[(i+1)%4] is less than the unit movement distance *d*.

Finding the next point:

There are many ways to calculate the next point. Note that the line segment from slug[i] to slug[(i+1)%4] forms the hypotenuse of a right-angle triangle, where the remaining sides are the *x* and *y* axes. If the distance between the two slugs is *distance*, then the ratio of *d* / *distance* will give you the ratio that you need to apply to both the *x* distance and the *y* distance. If you obtain the *y* distance as

```
slug[(i+1)%4].y – slug[i].y
```

and multiply it by the ratio, the result is the number of units that the slug[i] moves along the *y* axis. Note that a positive result moves the slug in a downward direction, while a negative result moves the slug in an upward direction.

Program Requirements:

Create a program named Slugs. It will implement the simulation described above.

Your program must:

- Read values using a `Scanner` object that opens the file called `slug_details.txt`. This file will contain a single text line with the following values in this order:
  1. The name of the file that will ultimately contain the line information (described below).
  2. The integer value of the sandbox size. Since it is a box, we only need one value.

3. The unit movement distance (*d*).

A sample `slug_details.txt` is provided. The values can be changed, but the order and layout must not. For testing purposes, d should begin with a larger number to reduce the number of iterations. The example picture at the top of this document was created with a boxSize of 400 and d =10.

- Implement the simulation as described, using the values provided from the input file.
- Use an array to store the Point values for each slug. **You may not use any of the Array type classes in the java API.**
- Print all the line segments to a text file whose name matches the name given in `slug_details.txt`.
    - The first line of the file must contain the length and width of the sandbox; since it's a square, is the value of boxSize written twice.
    - Subsequent lines must contain one line segment per line in the format given above, e.g. for the line (0, 200), (1, 175):
      0 200 1 175
    - The output file must be generated using a `PrintStream` object.
- Handle I/O exceptions by using a try / catch approach. Your code should print appropriate error messages for I/O exceptions.
- Use good procedural design (see text pp. 291—296 (ed. 3) or 283-288 (ed. 2) for ideas). Some examples of methods:
    - a method that calculates the distance between points,
    - a method that finds the next position of a slug

Getting started:

We recommend you start your assignment by following these steps:

1. Download the following files to your working directory:
    - DrawLines.java
    - slug_details.txt
2. Read this whole document.
3. Test the DrawLines program.
    - DrawLines.java is a helper file that does not need to be understood or changed. Its purpose is to render the lines contained in a formatted text file, so that you can see your output graphically!
    - Create a simple text file (called cross.txt) with the following information:

      200 200
      0 100 200 100
      100 0 100 200

    These values represent a rectangle of size 200 X 200 pixels, with a vertical line and horizontal line crossing in the middle. Now type the following on the console:

      ```
      javac DrawLines.java
      java DrawLines
      ```

    At the prompt, type `cross.txt`. You will see a small window with the drawing. Closing this window will stop the program.
4. Break down the programming task into parts. Build and test one part at a time.

Grading – The marker will look for:

- Documentation: Documentation in the implemented code must include the author's identification and the purpose of the program. Each group of instructions must be preceded with a comment describing their common purpose. Each method must be preceded with a comment describing its purpose as well as an indication of the input to and the output from the method.
- *White Space* and *Indentation* in the code and adherence to Java naming conventions
- Your program defines and uses methods wherever it is appropriate. Good procedural decomposition (i.e. appropriate breakdown of the code into logical methods) is expected.
- Compiles and produces correct output: The code constitutes a valid Java program (i.e. compiles *and* runs without error on the ECS 250 lab machines).
- Your program reads and uses the values in the input file and prints output to the output file EXACTLY as described above.
- Proper handling of I/O exceptions
- Proper use of arrays, without using the Array class.

More to Discover

The process of creating and testing a small part of the entire program, then adding on another small part and then another is called incremental development. It allows the programmer to simplify a large problem working through only a few issues at a time: http://www.upedu.org/references/bestprac/im_bp1.htm or http://en.wikipedia.org/wiki/Iterative_and_incremental_development.