CS100M Spring 2007 Project 4 due Thursday 3/29 at 6pm

Submit your three files `Calculator.java`, `Chaos.java`, `Fern.java` online in CMS before the project deadline. Be careful to submit your `.java` files and not the `.class` files. Both correctness and good programming style contribute to your project score.

> You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must register as a group in CMS and submit your work as a group.

Objectives

In this project, you will learn to write Java programs in the "procedural" style—the way we have written MATLAB programs up to this point. (So this is not object-oriented programming yet!) You will start with a simple program with one class and one method only to work with arithmetic operators, methods in the Math class, type conversion, and printing. In the second question, which has two subquestions, you will get to know two beautiful fractal objects—the Sierpiński triangle and Barnsley's Fern. Here you will deal with selection statements and loops. You will learn to generate non-uniform random distributions and you will write some static methods.

Do not use arrays in this project. In each class write the methods exactly as specified. Do not modify the provided code unless specified.

1. Calculator

This is the warm-up exercise. Write a program `Calculator.java` (the class name is Calculator) to perform the following operations and print the results. The output should be "labeled". For example, the output format for part (a) below may be `a: 1628`

a) (117+31)*11

b) 37/17

c) 37.0/17

d) The remainder of 74.23/6.28

e) The quotient of 34/13

f) The remainder of -87/9

g) Find the quotient (integer) for 74.23/6.28. Hint: use casting.

h) The result of evaluating "23 is greater than 11 and less than 45".

i) The result of evaluating "2^2 is not equal to 5".

j) Evaluate the expression "$(\sqrt{37.001})^2$ is equal to 37.001" (To do this you may use `Math.sqrt` but not `Math.pow`)

k) The bigger value between 2^{20} and 3^{12} (Use methods `Math.pow` and `Math.max`)

l) Evaluate the expression $\sin^2(\pi/4) + \cos^2(\pi/4)$. (Use `Math.sin`, `Math.cos`, and `Math.PI`)

m) Generate a random number on the interval $[-1, 12]$. (Use `Math.random`)

n) Randomly print one letter from the set {'a', 'b', 'c', 'd', 'e'}. The probability of each letter should be equal.) (Use `Math.random`)

2. Fractals

This question contains two subquestions. It might be helpful if you read both of them before you start coding.

a. Chaos Game

Start with an equilateral triangle T in the plane whose vertices are $v_1 = (0, 0)$, $v_2 = (1, 0)$, $v_3 = (1/2, \sqrt{3}/2)$. Then pick an arbitrary point P_0 in the plane and define a sequence of points $\{P_n\}$ as follows: Start at P_0; then choose one of the triangle's vertices at random and let P_1 be the point halfway between P_0 and the chosen vertex; then again randomly choose one of the triangle's vertices and let P_2 be the point halfway between P_1 and the chosen vertex. Continuing in this way yields an infinite sequence of points. Figure 1 illustrates the first few points of such a random sequence. We can think of this sequence as a particle moving along a random trajectory.

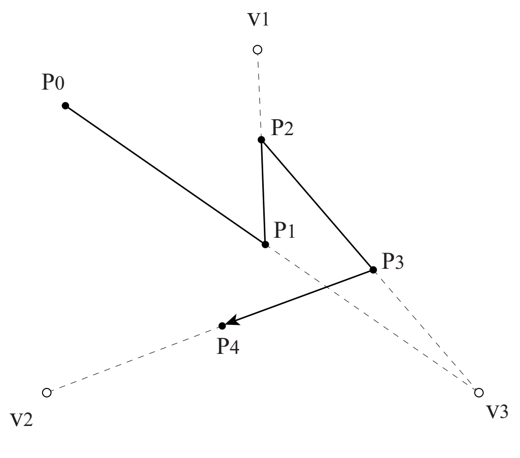

Figure 1. A possible trajectory in the chaos game.

This process is called the *chaos game* because of the apparently patternless movement of the point. When we plot the points on the screen, surprisingly, a clear pattern emerges, as shown in Figure 2. This figure is known as the Sierpiński triangle[1]. This question and the next will help

1. `http://mathworld.wolfram.com/SierpinskiSieve.html`

you appreciate the fact that simple mathematical manipulation of numbers can create somewhat artistic images.

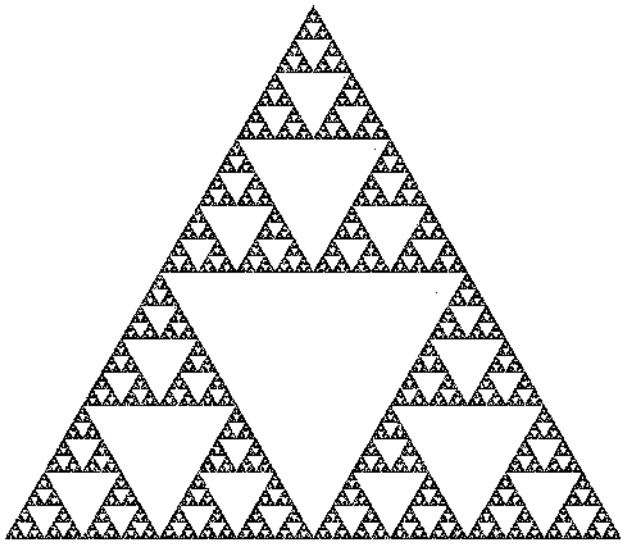
Figure 2. The Sierpiński triangle.

Your task is to generate the sequence of points $\{P_n\}$ and use them to plot the Sierpiński triangle. You only need to generate about 3^8 points in order to spot this pattern, but you are allowed to plot more. But you don't need to plot the vertices v_1, v_2 and v_3. You can use the function `Math.random` to get a random number uniformly distributed over the range $[0, 1)$.

You will write all your code in the `main` method of `Chaos.java`. You don't need to learn how to use the graphics functions of Java. A helper class `MyFrame` has been provided so that you can simply put `MyFrame.java` into the same directory as your project and use the statement

```
MyFrame f = new MyFrame("Chaos Game");
```

to get a handler of a graphics window titled "Chaos Game". Once you have a graphics window, you can use a statement like

```
f.drawPoint(x, y, Color.black);
```

to draw a point at the Cartesian coordinate (x, y). The default plot range of the window is the unit square $[0, 1] \times [0, 1]$, which is big enough for this question. The color names are defined in the class java.awt.Color, so you need to put this line at the beginning of your file

```
import java.awt.Color;
```

You can change the color to any color that is comfortable to your eyes, for example `Color.red`, `Color.orange`...

Submit your file `Chaos.java`.

b. Barnsley's Fern

In this question you will learn to generate non-uniformly distributed random numbers and utilize them to draw a lifelike image as in Figure 3. This figure was found by Barnsley[2] and it looks very much like a black spleenwort fern.

2. `http://mathworld.wolfram.com/BarnsleysFern.html`

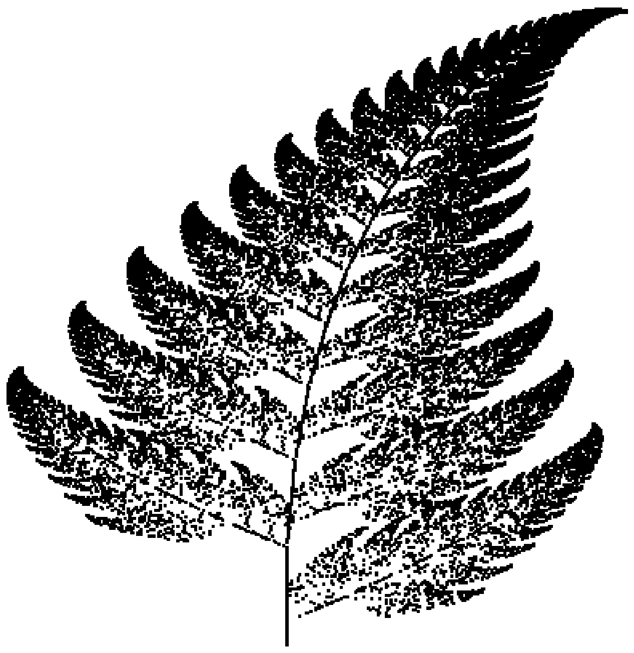

Figure 3. Barnsley's Fern.

Barnsley's Fern can be drawn in almost the same way as the Sierpiński triangle. Start from an arbitrary point; then randomly choose one of the following four functions to get the coordinate of the next point.

$$\begin{aligned} f_1(x,y) &= (0.85x + 0.04y, -0.04x + 0.85y + 1.6) \\ f_2(x,y) &= (-0.15x + 0.28y, 0.26x + 0.24y + 0.44) \\ f_3(x,y) &= (0.2x - 0.26y, 0.23x + 0.22y + 1.6) \\ f_4(x,y) &= (0, 0.16y) \end{aligned}$$

We call these functions the *fern functions*. Note that you need to update the point's coordinates simultaneously, or you might corrupt your results. If you are not sure how to code them, here is how you might code f_1 in Java

```
newx = 0.85*x + 0.04*y + 0;

newy = -0.04*x + 0.85*y + 1.6;

x = newx; y = newy;
```

But notice that we don't pick the four functions with the same probability. The probabilities of the four functions being chosen are 0.85, 0.07, 0.07, and 0.01. This means, one expects that in 100 applications, we have approximately applied f_1 85 times, f_2 7 times, f_3 7 times, and f_4 only once. How can we generate this non-uniform distribution? You can simply think of this like throwing a dart blindly into a segmented range [0,1) as shown in Figure 4. You decide which function to apply according to which region the dart ends up in. Obviously, the dart will lie in the range of [0, 0.85) more likely than in the range of [0.85, 1). If you toss it 100 times, one expects that it will end up in [0, 0.85) approximately 85 times and in [0.85, 1) only 15 times. Thus we have a non-uniform distribution of 0.85 and 0.15. You need to generalize this idea for your purpose.

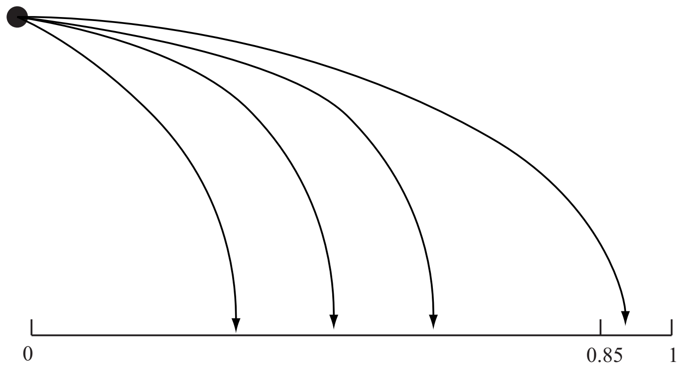

Figure 4. Generating non-uniform random numbers from uniformly distributed random numbers.

You will write two methods in `Fern.java`. First, write a static method `biasedFour` which takes no arguments. When called, the function returns one of the four possible numbers {1, 2, 3, 4} with probabilities 0.85, 0.07, 0.07, and 0.01 respectively. You must use the following function header

```
public static int biasedFour()
```

Then, in your `main` method, generate the sequence of points for Barnsley's fern and plot them. When generating each point, call `biasedFour` to determine which of the four fern functions $\{f_1, f_2, f_3, f_4\}$ to use.

You will use the same graphics helper class `MyFrame` as in Question 2(a) and use the function `f.drawPoint` to draw the points. After you get a handler to the graphics window, you need to change the range of the graphics window to the region so that x ranges from -3 to 3 and y ranges from -1 to 11. You can use the following code to set the window to the required range.

```
f.setRange(-3, 3, -1, 11);
```

Again, please remember to use `import java.awt.Color;` at the beginning of your file, or you won't be able to use the color names.

Submit your file `Fern.java`.