Homework 7

The purpose of this assignment is gaining practice in recursive programming. Therefore, in each exercise, you must provide a recursive solution; other solutions will not be accepted. And, you may not use loops at all. Your implementation may or may not include private functions, as you see fit.

1. Hailstone Sequences, Revisited

In homework 3 we described how to generate a *hailstone sequence*: Start with a positive integer which we call the *seed*, and obtain a sequence of numbers by following these rules: (i) If the current number is even, divide it by 2; otherwise, multiply by 3 and add 1; (ii) Repeat. The sequence terminates when (and if) it reaches the number 1.

Write a function with the following signature: `public static void hailstone(int seed).` The function prints the hailstone sequence that starts with the given seed. For example, `hailstone(6)` prints the sequence `6 3 10 5 16 8 4 2 1`, and `hailstone(16)` prints the output `16 8 4 2 1.`

Note: If the *Collatz conjecture* is true, then executing the `hailstone` function with any seed is guaranteed to stop at some point or another.

2. Decimal To Binary Conversion

The function `integerToBinary` returns the binary representation of a given non-negative integer, as a string. For example, `integerRoBinary(6)` returns `"110"`, and `integerRoBinary(16)` returns `"10000"`.

Write the function, using the signature `public static String integerToBinary(int x).`

Implementation tips: Each step of the recursion should handle one binary digit. Pay attention to the order of operations in your implementation logic.

3. Parity

A *binary string* is a string that contains only `'0'` and `'1'` characters. The parity of a binary string is defined as follows. If the number of times that the character `'1'` appears in this string is even, the parity it 0; if it's odd, the parity is 1. For example, the parity of `"101"` is 0, the parity of `"10110"` is 1, and the parity of `"001001101"` is 0. Write a function, using the signature `public static int parity(String binaryStr).`

4. Increasing Subsequence

A sequence of numbers is said to be *monotonically increasing* (or simply *increasing*) if every number in the sequence is greater than, or equals to, the number preceding it. For example, 2, 8 ,8 ,11 is an increasing sequence. A sequence of numbers is said to contain an increasing subsequence if at least one of its sub-sequences is an increasing sequence. For example, the sequence 6, 4, 7, 0, 2, -4, 9, -6, contains a subsequence of length 3.

The boolean function `increasing(int[] x, int length)` returns `true` if the given array contains an increasing subsequence of the given length, and `false` otherwise. For example,

`increasing({9,2,4,8,9,1,2,3,9,2}, 4)` returns `true`,
`increasing({2,9,8,3,-2,2,-4,-5,-6,2}, 3)` returns `true`,
`increasing({{10,9,8,7,6,5,4,3}, 2)` returns `false`,
`increasing({9,7,5,4,7,1,-3,8}, 3))` returns `true`.

These examples are not exactly Java, but you got the picture. Write the function, using the signature `public static boolean increasing(int[], int length)`.

5. Banach Curve

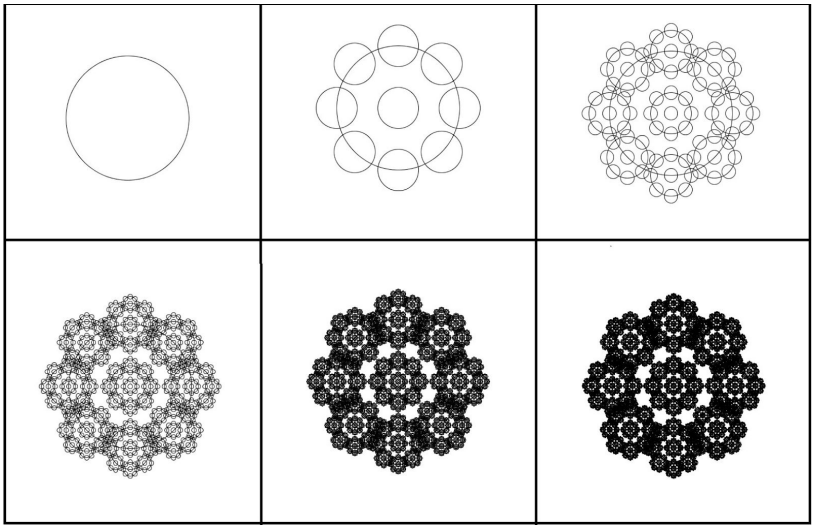
The so-called *Banach Curve* can be generated using the following fractal rule:

a. Draw a circle.

b. Draw 9 smaller circles, each with a radius ⅓ of the original circle. One of the smaller circles should have the same center as that of the original circle. The centers of the remaining 8 smaller circles should be equally spaced along the circumference of the original circle.

c. Repeat step b with each of the smaller circles.

For example, here is an illustration of the first 6 levels of this process:



Note: The circle of radius r centered at point (x, y) is the set of all the points $(x + r \cdot cos(t), y + r \cdot sin(t))$ where $0 \le t \le 2\pi$, and t is given in radians.

Write a function that draws a Banach curve of a given level n, using the signature `public static void banachCurve(int n)`.

Implementation tips:

(i) In addition to the function just described, we suggest using an auxiliary function with the signature `private static void banachCurve(double x, double y, double r, int n)`.

(ii) You may want to use the geometric insight described above to help compute the coordinates at which the smaller circles should be centered. You may also find Java's `Math.toRadians` function

useful. This function takes an angle, measured in degrees, and returns the angle, measured in radians. Both input and output degrees are represented as `double` values.

(iii) Use the `StdDraw` class for drawing the circles.

Submission

In this homework we provide no class skeleton. Create a new Java class, named `Recursion`, and write all five functions in it. Write clear `/** API documentation */` at the beginning of each function. Decide for yourself how to test all the functions. The graders should be able to easily follow, execute, and modify your tests.

Before submitting your solution, inspect your code and make sure that it is written according to our **Java Coding Style Guidelines**. Also, make sure that each program starts with the program header described in the **Homework Submission Guidelines**. Both documents can be found in the Moodle site, it is your responsibility to read them. Any deviations from these guidelines will result in points penalty. Submit the following file only:

- `Recursion.java`

Deadline: Submit Homework 7 no later than December 16, 23:55. You are welcome to submit earlier.