**Homework 7**

* Due Sunday by 11:59pm
* Points 20
* Submitting a text entry box or a file upload

This week we are going to implement two versions of the Fibonacci sequence and use threads to determine how much faster the second method is.

For those unfamiliar, the **Fibonacci** sequence of numbers is: 0 1 1 2 3 5 8 13 21

That is, starting with 0 and 1, every number after that is the previous two added together. What we want to do is write a couple of threads that find the n'th number of the Fibonacci sequence. So if n = 0, it would return 0, if n = 6, it would return 8, if n = 8 it would return 21.

The first solution uses **recursion** and is the simplest. In pseudocode it is:

1. int fibonacci(int n)
   1. if(n == 0) return 0
   2. if(n == 1) return 1
   3. return fibonacci(n-1) + fibonacci(n-2)

Very simple. Too simple as we will find out. The second method uses **dynamic** programming to store the previous calculations rather than re-calculating them. This one looks like:

1. int fibonacci(int n)
   1. v1 = 0, v2 = 1, v3 = 0
   2. for(i=2 to i <= n)
      1. v3 = v1 + v2
      2. v1 = v2
      3. v2 = v3

**Objective:** Implement both methods of solving the Fibonacci sequence into two separate classes that extend the Thread class. The threads will have some way to set n before the thread is started. In the run() function, you will execute the algorithm and time how long it takes to get an answer. At the end, the thread will output the answer to the screen along with the number of milliseconds it took.

**Hint**: To find milliseconds just call System.currentTimeMillis(); You want to capture this time BEFORE your thread starts searching for the answer then AFTER. What you output is the difference between those two variables. The timing may vary between runs, but with n=40, you should see a pretty large difference.

Attached is an example of my output along with what the 40th number of the Fibonacci sequence is. Note, it is fine if the dynamic algorithm takes 0ms, as that means it took less than 1 ms to complete (microseconds likely). If your timing is working correctly I would not expect the recursive solution to take less than 1ms

