COSC 3P91

Lab 7

In this lab you are going to use multiple threads and the **Producer-Consumer** design pattern.

1. Implement a generic class `Queue<E>` of a queue storing elements from `E`. This class is supposed to be thread safe, i.e., can be used safely in a multiple thread environment. The class has methods `void push(E element)` and `E pull()` for adding and getting the next element from the queue. The `pull()` method is supposed to be blocking, i.e., if `pull()` is called and the queue is empty, then the thread calling `pull()` is supposed to wait until an element becomes available. In addition, the class has to have a method `close()` that if called sets the queue in an inactive state. An inactive queue will ignore any call of `push()`. Any subsequent call of `pull()` on an inactive queue will continue to return the elements in the queue until it is empty. If empty, `pull()` will return `null` as an indication that there will be no more elements.
2. Implement a class `FibMaker` that is a thread. The class has an `int limit` and a `Queue<Integer> queue` that are passed in the constructor. In the `run()` method the `FibMaker` will send all Fibonacci numbers in ascending order smaller or equal to the `limit` to the queue. You can use a loop and two variables `int a` always containing the n-th Fibonacci number and `int b` always containing the n+1-st Fibonacci number to achieve this. Once the `limit` is reached the queue must be closed and the thread terminates.
3. Implement a class `PrimeMaker` that is a thread. The class has an `int limit` and a `Queue<Integer> queue` that are passed in the constructor. In the `run()` method the `PrimeMaker` will send all prime numbers in ascending order smaller or equal to the `limit` to the queue. You can use the sieve of Eratosthenes to achieve this. Once the `limit` is reached the queue must be closed and the thread terminates.
4. Implement a generic class `Filter<E>` that is `Runnable` for a generic class `E` extending `Comparable`. This class has two `Queue<E>`'s `queue1` and `queue2` that are passed in the constructor. In the `run()` method you should print any element that you receive from both queues, i.e., you find the same element in both queues,

to `System.out`. It is assumed that the elements in both queues are ascending (cf. Week 3, Slide 31). If there are no more elements, then print an appropriate message to `System.out`. and terminate.

5. In the `main()` method of the program create two queues, a `FibMaker` and a `PrimeMaker` with some `limit` and one of the queues, respectively. Furthermore, connect the two threads to an object of the `Filter` class by passing the queues. Finally, start the `FibMaker` and the `PrimeMaker` thread and call `run()` of the `Filter`.