handout #11

**CSE143—Computer Programming II**

**Programming Assignment #4**

**due: Thursday, 4/25/13**

This assignment will give you practice with queues. You are going to implement a class that computes all the primes up to some integer n.  The technique you are to use was developed by a Greek named Eratosthenes who lived in the third century BC.  The technique is known as the Sieve of Eratosthenes.

The algorithm is described by the following pseudocode:

create a queue and fill it with the consecutive integers 2 through n inclusive.  
create an empty queue to store primes.  
do {  
    obtain the next prime p by removing the first value in the queue of numbers.  
    put p into the queue of primes.  
    go through the queue of numbers, eliminating numbers divisible by p.  
} while (p < sqrt(n))  
all remaining values in numbers queue are prime, so transfer them to primes queue

You are to use the Queue interface discussed in lecture.  When you want to construct a Queue object, you should make it of type LinkedQueue.  These classes will be included in the zip file for the assignment.

You should define a class called Sieve with the following public methods:

|  |  |
| --- | --- |
| Method | Description |
| Sieve() | Constructs a sieve object. |
| void computeTo(int n) | This is the method that should implement the sieve algorithm.  All prime computations must be implemented using this algorithm. The method should compute all primes up to and including n.  It should throw an IllegalArgumentException if n is less than 2. |
| void reportResults() | This method should report the primes to System.out.  It should throw an IllegalStateException if no legal call has been made yet on the computeTo method.  It is okay for it to have an extra space at the end of each line. |
| int getMax() | This is a convenience method that will let the client find out the value of n that was used the last time computeTo was called.  It should throw an IllegalStateException if no legal call has been made yet on the computeTo method. |
| int getCount() | This method should return the number of primes that were found on the last call on computeTo.  It should throw anIllegalStateException if no legal call has been made yet on thecomputeTo method. |

Your reportResults method should print the maximum n used and should then show a list of the primes, 12 per line with a space after each prime.  Notice that there is no guarantee that the number of primes will be a multiple of 12.  The calls on reportResults must exactly reproduce the format of the sample log.  The final line of output that appears in the log reporting the percentage of primes is generated by the main program, not by the call on reportResults.

Even though you will be using a LinkedQueue object to write this program, you should use variables of type Queue.  It should be possible for someone to replace all calls on the LinkedQueue constructor with calls on the constructor of another class that implements the Queue interface and the rest of your code should work without modification.  You should also be sure to specify that we want queues of “Integer” (Queue<Integer> and LinkedQueue<Integer>).  If you fail to specify this properly, the compiler will warn you that you are using “unsafe” operations.

You must guarantee that your object is never in a corrupt state.  For example, your sieve object might be asked to compute up to one value of n and then asked to compute up to a different value of n without a call on reportResults ever being made.  Similarly, your object might be asked to compute up to some value of n and then be asked to reportResults more than once.  Each call on reportResults, getMax and getCount should behave appropriately given the previous call on computeTo, no matter how often they are called or in what order.  Finally, notice that if reportResults, getMax or getCount are called before a legal call on computeTo, they throw an exception to indicate that the operation is not legal given the object’s state.

In terms of correctness, your class must provide all of the functionality described above.  In terms of style, we will be grading on your use of comments, good variable names, consistent indentation and good coding style to implement these operations.  Remember that you will lose points if you declare unnecessary data fields (redundant fields or variables that can be local to a method).

You should name your file Sieve.java and you should turn it in electronically.

The zip file for this assignment (ass4.zip) includes a file SieveMain.java that constructs a sieve object and makes calls on it based on values entered by the user.  You can use this program to test your class, but keep in mind that it does not test the internal consistency of your object.  You will need to have Queue.java and LinkedQueue.java in the same directory as your Sieve.java in order to run SieveMain.

The zip file includes sieve.txt, a longer sample log of execution than the one below.  The longer log is also available with the output comparison tool which you can use to check your program’s output.

**Log of Execution (user responses underlined)**

This program computes all prime numbers up to a

maximum using the Sieve of Eratosthenes.

Maximum n to compute (0 to quit)? 20

Primes up to 20 are as follows:

2 3 5 7 11 13 17 19

% of primes = 40

Maximum n to compute (0 to quit)? 100

Primes up to 100 are as follows:

2 3 5 7 11 13 17 19 23 29 31 37

41 43 47 53 59 61 67 71 73 79 83 89

97

% of primes = 25

Maximum n to compute (0 to quit)? 0