CMPS 12A
Introduction to Programming
Spring 2010

Programming Assignment 4

Due Friday May 14, 10:00 pm

In this project you will write a Java program that reads a positive integer *n* from standard input, then prints out the first *n* prime numbers. We say that an integer *m* is *divisible* by a non-zero integer *d* if there exists an integer *k* such that $m = k \cdot d$, i.e. if *d* divides evenly into *m*. Equivalently, *m* is divisible by *d* if the remainder of *m* upon (integer) division by *d* is zero. We would also express this by saying that *d* is a *divisor* of *m*. A positive integer *p* is called *prime* if its only positive divisors are 1 and *p*. The one exception to this rule is the number 1 itself, which is considered to be non-prime. A positive integer that is not prime is called *composite*. Euclid showed that there are infinitely many prime numbers. The prime and composite sequences begin as follows:

Primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, …
Composites: 1, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, …

There are many ways to test a number for primality, but perhaps the simplest is to simply do trial divisions. Begin by dividing *m* by 2, and if it divides evenly, then *m* is not prime. Otherwise, divide by 3, then 4, then 5, etc. If at any point *m* is found to be divisible by a number *d* in the range $2 \le d \le m-1$, then halt, and conclude that *m* is composite. Otherwise, conclude that *m* is prime. A moment's thought shows that one need not do any trial divisions by numbers *d* which are themselves composite. For instance, if a trial division by 2 fails (i.e. has non-zero remainder, so *m* is odd), then a trial division by 4, 6, or 8, or any even number, must also fail. Thus to test a number *m* for primality, one need only do trial divisions by *prime* numbers less than *m*. Furthermore, it is not necessary to go all the way up to $m-1$. One need only do trial divisions of *m* by primes *p* in the range $2 \le p \le \sqrt{m}$. To see this, suppose $m > 1$ is composite. Then there exist positive integers *a* and *b* such that $1 < a < m$, $1 < b < m$, and $m = a \cdot b$. But if both $a > \sqrt{m}$ and $b > \sqrt{m}$, then $a \cdot b > m$, contradicting that $m = a \cdot b$. Hence one of *a* or *b* must be less than or equal to $\sqrt{m}$.

To implement this process in java you will write a function called `isPrime()` with the following signature:

```
static boolean isPrime(int m, int[] P)
```

This function will return `true` or `false` according to whether *m* is prime or composite. The array argument *P* will contain a sufficient number of primes to do the testing. Specifically, at the time `isPrime()` is called, array *P* must contain (at least) all primes *p* in the range $2 \le p \le \sqrt{m}$. For instance, to test $m = 53$ for primality, one must do successive trial divisions by 2, 3, 5, and 7. We go no further since $11 > \sqrt{53}$. Thus a precondition for the function call `isPrime(53, P)` is that $P[0] = 2$, $P[1] = 3$, $P[2] = 5$, and $P[3] = 7$. The return value in this case would be `true` since all these divisions fail. Similarly to test $m = 143$, one must do trial divisions by 2, 3, 5, 7, and 11 (since $13 > \sqrt{143}$). The precondition for the function call `isPrime(143, P)` is therefore $P[0] = 2$, $P[1] = 3$, $P[2] = 5$, $P[3] = 7$, and $P[4] = 11$. The return value in this case would be `false` since 11 divides 143. Function `isPrime()` should contain a loop that steps through array *P*, doing trial divisions. This loop should terminate when

either a trial division succeeds, in which case `false` is returned, or until the next prime in P is greater than $\sqrt{m}$, in which case `true` is returned.

Function main() in this project will read the command line argument *n*, allocate an int array of length *n*, fill the array with primes, then print the contents of the array to stdout according to the format described below. In the context of function main(), we will refer to this array as `Primes[]`. Thus array `Primes[]` plays a dual role in this project. On the one hand, it is used to collect, store, and print the output data. On the other hand, it is passed to function `isPrime()` to test new integers for primality. Whenever `isPrime()` returns `true`, the newly discovered prime will be placed at the appropriate position in array `Primes[]`. This process works since, as explained above, the primes needed to test an integer *m* range only up to $\sqrt{m}$, and all of these primes (and more) will already be stored in array `Primes[]` when *m* is tested. Of course it will be necessary to initialize $\text{Primes}[0] = 2$ manually, then proceed to test 3, 4, … for primality using function `isPrime()`.

The following is an outline of the steps to be performed in function main().

- Check that the user supplied exactly one command line argument which can be interpreted as a positive integer *n*. If the command line argument is not a single positive integer, your program will print a usage message as specified in the examples below, then exit.
- Allocate array `Primes[]` of length *n* and initialize $\text{Primes}[0] = 2$.
- Enter a loop which will discover subsequent primes and store them as $\text{Primes}[1]$, $\text{Primes}[2]$, $\text{Primes}[3]$, ……, $\text{Primes}[n-1]$. This loop should contain an inner loop which walks through successive integers and tests them for primality by calling function `isPrime()` with appropriate arguments.
- Print the contents of array `Primes[]` to stdout, 10 to a line separated by single spaces. In other words $\text{Primes}[0]$ through $\text{Primes}[9]$ will go on line 1, $\text{Primes}[10]$ though $\text{Primes}[19]$ will go on line 2, and so on. Note that if *n* is not a multiple of 10, then the last line of output will contain fewer than 10 primes.

Your program, which will be called Prime.java, will produce output identical to that of the sample runs below. (As usual % signifies the unix prompt.)

```
% java Prime
Usage: java Prime [PositiveInteger]
% java Prime xyz
Usage: java Prime [PositiveInteger]
% java Prime 10 20
Usage: java Prime [PositiveInteger]
% java Prime 75
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379
%
```

As you can see, inappropriate command line argument(s) generate a usage message which is similar to that of many unix commands. (Try doing the more command with no arguments to see such a message.) Your program will include a function called `Usage()` having signature

```
static void Usage()
```

that prints this message to stderr, then exits. Thus your program will contain three functions in all: `main()`, `isPrime()`, and `Usage()`. Each should be preceded by a comment block giving it's name, a short description of it's operation, and any necessary preconditions (such as those for `isPrime()`.) See examples on the webpage.

What to turn in

Submit the file Prime.java to the assignment name pa4 in the usual way:

```
% submit cmps012a-pt.s10 pa4 Prime.java
```

This project is significantly more complex than previous assignments, so get started early and ask questions if anything is less than clear.