This assignment is due on Friday, October 10. You should work on this programming assignment on your own, without discussing it with other people in the class.

You are not required to include comments in your programs. With no comments, my program for Exercise 1 is 40 lines long, and my program for Exercise 2 is 80 lines. Place copies of your completed programs in the directory /classes/f08/cs229/zz9999, where you should replace "zz9999" with your own username. I will compile, run, and print out the programs myself. You do not have to turn in printouts.

1. (**Simple Spell Checker.**) Java has a standard class named *HashSet* that can be used to represent sets. (It is defined in package `java.util`.) *HashSet* is a parameterized class, so that you should use it in a form such as *HashSet<String>*, for example, to represent sets of values of type *String*. The constructor "`new HashSet<String>`" creates a set of *Strings* that is initially empty. If *set* is an object of type *HashSet<String>*, then you can use the following methods: `set.add(str)` adds a *String*, *str*, to the set if it is not already there; `set.size()` returns the number of elements in the set; and `set.contains(str)` returns *true* or *false* to indicate whether or not *str* is an element of the set.

The file `/classes/f08/cs229/wordlist` contains a long list of words, one per line. Write a program that reads all the words from this file and stores them in a *HashSet*. The program should then enter a loop in which it reads a word from the user and checks whether the user's word is in the set. I suggest converting all words to lower case, both words from the file and words from the user. (The program actually implements a simple spell check. The file is assumed to contain a list of correctly spelled words, so that when the program checks whether the user's word is in the set, it is actually checking whether the word is spelled correctly.)

For input, you can use either the built in class *Scanner* from package `java.util`, or the non-standard *TextIO* class. (There is a copy of TextIO.java in the directory */classes/f08/cs229*.) In case you don't know how to read a file using *Scanner*, here is some code that will do it:

```
Scanner file;
try {
    file = new Scanner(new File("/classes/f08/cs229/wordlist"));
}
catch (Exception e) {
    System.out.println("Can't find dictionary file.");
    return;
}
while (file.hasNextLine()) {
    String line = file.nextLine(); // get one line from the file.
    // insert code to process the line here
}
file.close();
```

(If you are **really** ambitious—and possibly would like to get a little extra credit—you can try to write a more capable spell checker that provides a list of alternatives for misspelled words entered by the user. The idea is make small modifications of the user's input and look them up in the word list. Any modification that is found in the word list is placed

on the list of alternatives that will be displayed to the user. They types of modifications that are useful include: drop one of the letters from the user's input; change one of the letters in the user's input to another letter; insert a letter into the user's input; and swap the order of two consecutive letters in the user's input. Of course, you should attempt to apply each modification in all possible ways. For example, for substitution modification, you should go to *each* position in the user's input and replace the letter there with *each* of the 26 letters in the alphabet. To do any of this, you will need to work with substrings. Look up the *substring* method in the *String* class.)

2. (**Search for Primes**.) The *Sieve of Erastothenes* is an algorithm for finding every prime number that is less than or equal to some specified upper bound, M. To apply the sieve, place all the numbers between 2 and M, inclusive, into a set, A. Then, apply the following procedure:

```
for i from 2 to M
    if i is in the set A
        remove all multiples of i, starting with 2*i, from A
```

At the end of this procedure, A contains all the primes, and only the primes, in the range 2 through M. To apply the sieve of Erastothenes for a large value of M, you need an efficient implementation of sets of integers. The standard Java class *BitSet*, which is defined in the package `java.util`, can efficiently represent sets of non-negative integers. For an integer n, the constructor `new BitSet(n)` creates an object that can represent any subset of the set $\{0, 1, 2, \ldots, n-1\}$ using just one bit for each potential member of the set. If *numbers* is such an object then the methods that are defined for *numbers* include:

numbers.set(i) — add i to the set (if it is not already there).

numbers.clear(i) — remove i from the set (if it is there).

numbers.get(i) — test whether or not i is an element of the set.

Write a Java program that uses a *BitSet* and the Sieve of Eratosthenes to find the number of primes in the range 2 through M, where M is an integer input by the user. Furthermore, once you have the set of primes in this range, use it to answer questions of the form: Given a number k in the range 1 through M, what is the next number greater than or equal to k and less than or equal to M that is prime? (It is possible that the answer is "none" if there is no prime number between k and M.) Your program should first ask the user for the upper limit, M. It should then run the Sieve of Eratosthenes. Next, it should print out the number of primes in the range 1 through M. Finally, it should go into a loop where it gets a value for k from the user and outputs the answer to the above question. You can end the program when the user enters a value of 0 for k. I suggest that you try your program for small values of M, and also run it with $M = 100,000,000$. For input from the user, you can use the built-in *Scanner* class, or you can use the non-standard *TextIO* class, whichever you are used to.

Note that there are additional methods in the *BitSet* class, in addition to those listed above, that you might find useful. Check out the API.

(Your program would also work for $M = 1,000,000,000$, if you want to try it for that value, but for that large a set, it will need more memory than Java usually gets when it runs. To request the extra memory that you need, run the program with the command `java -Xmx150m Sieve` (where *Sieve* is the name of the program). It will take some time for the sieve to complete on so large a value; my program took two minutes.)