Lab 4

Due March 6

Handout 5
CSCI 136: Spring 2017
February 27

Comparators Are My Favorite Sort of Vectors

1 Important Notes

- This week, **you may work with a partner**. If you choose to do so, you should only submit one lab with both names clearly indicated in comments at the top.
- As in prior labs, you should develop a design document before coming to lab. We recommend that you and your partner each create independent design documents, then discuss your two appraoches together to finalize a design.

2 Questions

Answer these questions in a file called README and submit it with the rest of your code this week. I encourage you to answer these questions **before** lab, although it is not necessary to submit anything before the due date. **Note: These are the Regular Problems— not the Self-Check Problems!**

Problem 5.5
Problem 5.23
Problem 5.26
Problem 6.3 (Refer to bubbleSort implementation in text.)

Problem 6.4

3 Lab

The goals of this week's lab are to gain experience with

1. Java's `Comparator` interface, and
2. sorting by various criteria.

In class, we saw how to implement the `Comparable` interface and write a `compareTo` method to compare to objects of the same class. While this is sufficient in many cases, we sometimes will want to sort the same data in different ways. For example, we may wish to sort `Student` records by name in some situations and by grade point average in other situations. A single `compareTo` method is not sufficient to do this, because `compareTo` only provides one way to sort such records.

In order to sort objects in multiple ways, the `structure5` package uses `Comparators`. A `Comparator` object's sole purpose is to compare two objects and return a value indicating which one is smaller. We can then sort in different ways by using different kinds of `Comparator` objects in the sorting algorithm. See Chapter 6.8–6.9 for a discussion of `Comparators` and how to use them.

In this lab we will develop an extension of `Vector`, called `MyVector`, that includes a new method `sort` that will (with the help of a `Comparator`) order the elements of the `Vector`. Here are the basic steps for implementing this new class:

1. Before starting, copy the following starter directory (don't forget the period in this line):

   ```
   cp /opt/mac-cs-local/share/cs136/labs/vector/* .
   ```

   This directory contains starter files for `MyVector.java` and `Student.java`, plus `newphonebook.txt` for the last part.

2. Create a new class, `MyVector`, which is declared to be an extension of the `structure5.Vector` class. Since we are using generic structures, the class header for `MyVector` will be:

   ```
   public class MyVector<E> extends Vector<E>
   ```

   You should write a default constructor for this class that simply calls `super();`. This will force the super-class constructor in Vector to be called. This, in turn, will initialize the protected fields of the parent class.

3. Construct a new method called `sort` . It should have the following declaration:

   ```
   // pre: c is a valid comparator
   // post: sort this vector in order determined by c
   public void sort(Comparator<E> c)
   ```

   This method uses the `Comparator` object to actually perform a sort of the values in `MyVector`. You may use any sort algorithm, although it may be good to start with a straightforward approach, such as bubble sort.

   The class `Comparator` is parameterized by the type of object that it can compare:

   ```
   public interface Comparator<T> {
     /*
      * Returns:  < 0  if a is smaller than b
      *           0    if a equals b
      *           > 0  if a is larger than b
      */
     int compare(T a, T b);
   }
   ```

   In the case of the `sort` method, the type of `c` is `Comparator<E>`. That is, the `c` object must be a comparator for the type of data stored in the vector.

   When writing new comparators, you will specify what type they are defined for. For example, we would define a `CardComparator` class to compare `Card` objects as follows:

   ```
   import java.util.Comparator;

   public class CardComparator implements Comparator<Card> {
      ...
   }
   ```

   The `compare` method in that class would then take two `Card` objects as parameters.

   Be sure to test `MyVector` thoroughly before going on to the next part. `MyVector` inherits a `toString` method from `Vector`, which should be handy for printing out the contents of your vectors during testing.

4. You are now going to write a program that reads Williams College phone book data into a Vector and then answers some questions by sorting it with `Comparator`s applied to the student entries. The file `newphonebook.txt` contains student entries, represented by three lines, and separated by a line of dashes:

```
Iluv C Science
Poker Flats B5
4135973427 3334 5406394821
-----------------
Jeannie Albrecht
Thompson Chemistry Lab 304
4135974251 1234 4134581234
-----------------
...
```

The first line is the name of the student, the second is their campus address, and the third contains the campus phone, su box, and home (or cell) phone. You should create a `Student` class which represents all the information for a single student.

Read in the data file and create a `MyVector` of `Student` objects. You should read in the phone numbers as `long`s (with the `Scanner`'s `nextLong()` method) rather than `int`s, because integer variables cannot store numbers greater than about 2 billion due to how they are represented inside the computer. You may wish to create a new class that will be responsible for reading in the data and performing the operations below.

Your program should then print out answers to **at least four** of the following questions. (You can pick which four you want to implement.)

(a) Which student appears first in a printed phone book, if names are printed as they appear in the data file (ie, first name first)?

(b) Which student has the smallest SU box? Largest?

(c) Which student has the most vowels in his or her full name? (You may ignore "Y"s when counting vowels.)

(d) Which address is shared by the most students, and what are their names? You may find it useful to build a second vector storing `Associations` between each address and the number of students living there. A special comparator can then be used to sort that vector by comparing the number of students at each address. Once the most common address is known, you can consult the original vector of `Student`s and print those living at that address.

Some students have address `UNKNOWN` because they are abroad, on leave, etc. These students should be ignored for this question. Any other student entries with strange formatting, should also be ignored. (But please let your instructor know if you find any weird entries.)

(e) What are the ten most common area codes for student home phone numbers, in decreasing order? Some phone numbers are -1 to indicate that the actual information is not available. You should simply disregard students without a known home phone number.

(CONTINUED ON NEXT PAGE)

4 Deliverables

When you have completed the lab, place your files in a directory called `<unix>-lab4` (with `<unix>` replaced by your Williams Unix), and submit it in the drop-off folder for your lab section. Your `<unix>-lab4` directory should include the following:

- Your well-documented source code for all Java files used.
- A `README` file that includes the following:
  1. answers to the 5 questions from Section 2,
  2. a high-level description of what is in each Java file,
  3. the answers (printed by your program) to at least four questions at the end of Section 3.

As in all labs, you will be graded on design, documentation, style, and correctness. Be sure to document your program with appropriate comments, including a general description at the top of the file, a description of each method with pre- and post-conditions where appropriate. Also use comments and descriptive variable names to clarify sections of the code which may not be clear to someone trying to understand it. If you worked with a partner, please submit one version of the lab with both of your names in the README and each of the .java files.