Programming Project 5 - Due: Sunday, May 22 at 11:59 PM

Satisfied with the work you did finding SMC alumni at UC schools, your boss has come to ask for your help again. She has a large file of students she needs some help with.

Part 1:

It would be helpful to her to have the students sorted by the school they are attending and students attending the same school sorted by ID in ascending order. For example:

Joe Paarmann, 3,UCB
Otha Baloy,5,UCB
...
Alton Hamblet,1,UCD
Jessie Merle,7,UCD
Lawanda Doell,9,UCD
...
Alva Zajicek,4,UCI
...
Chester Kemfort,2,UCLA
Alice Mines,6,UCLA
...

She tried sorting the students herself but when she sorts by school the students are no longer sorted by ID within each school, and vice-versa. Having learned of stable sorting you agree to take on this simple task. **You will use the Java library sort to do the sorting.**

a) Since the sort is stable, you can do **two consecutive sorts** and end up with the needed result.

To sort a list by some criteria in Java 7, you use the Collections.sort() method like below. The second argument is an implementation of the Comparator interface, which determines how two elements are compared.

```
Collections.sort(myList, myComparator);
```

b) It occurs to you that you don't need two separate sorts. You plan to write a single comparator that sorts by school and breaks ties by ID and use that to sort only once to get the desired result.

Part 2:

Looking at the data you notice that some first names are pretty popular while others pretty rare. You plan to write some code that will give you a list (containing no duplicates) of all the **first names** which occur a given number of times or more (e.g. all the first names occurring 50 times or more). You can only store the data in a list. No other ADTs are allowed.

Part 3:

Having learned of the Map ADT, you think that would be appropriate to use in getting the common names. You will solve the problem in Part 2 again, but now you will use the Map ADT. Have a look at the TreeMap class documentation. Don't be intimidated by the large number of methods; you will only need to use the main ones we have seen.
https://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html

Part 4:

In the report file you will do a brief (2 lines each) complexity analysis of Parts 2 and 3 and give the big O for each.

Deliverables:

You should submit a zip file named project5_first_last.zip (where first and last are your first and last name) containing **ONLY the files below.**

Project5.java

report.txt : a text (not Word, Power Point, ...) file containing ONLY:
a) A 1 to 5 lines paragraph from you saying "I have tested this program and there are no known issues." if you believe that to be the case, or a brief description of known issues in case your program has known problems or you could not fully implement it.
b) The complexity analysis for Part 2 and 3

How you get points:

- Part 1 20 points
- Part 2 35 points
- Part 3 35 points
- Part 4 10 points

How you lose points:

- If you do not follow the given directions and decide to make changes "for fun". Specifically, **do not change the code given to you**. **Just fill in the missing method implementations.**
- If your implementation is inefficient. Your solution should be efficient to a level seen in class for this ADT.
- If your code does not follow good coding practices.
- If any of your code prints anything at all on the console. Remove all your print outs, debug statements, etc. Clean up your code and do not leave clutter behind.
- If your code has no comments where needed. Comment your code appropriately. Brief and to the point.
- If you submit your whole workspace or executable files. Submit only the files the project asks for.