Project 6

Due Date: Wednesday 27 April by 11:59 PM

General Guidelines.

The method signatures provided in the skeleton code indicate the required methods. You may need additional methods or classes that will not be directly tested but may be necessary to complete the assignment, and you are welcome to add those into the classes.

Unless otherwise stated in this handout, you are welcome to add to/alter any provided java files as well as create new java files as needed, but make sure that your code works with the provided test cases. Your solution must be coded in Java. Also, keep in mind that your solution should not depend on any changes you make to any files that are not submitted. For example, the test code is not submitted as we use our own test code to grade your project, so if your solution depends on changes you made to the test code, then it will not pass our test code. Again: Only changes made to submitted files will be reflected in the grading, and we do not want you to submit anything that is not specifically mentioned in the submission guidelines.

In general, you are not allowed to import any additional classes in your code without explicit permission from your instructor! The Math class is fine for import.

Note on academic dishonesty: Please note that it is considered academic dishonesty to read anyone else's solution code, whether it is another student's code, code from a textbook, or something you found online. You MUST do your own work! It is also considered academic dishonesty to share your code with another student. Anyone who is found to have violated this policy will be subject to consequences according to the syllabus and university policy.

Note on grading and provided tests: The provided tests are to help you as you implement the project and (in the case of auto-graded sections) to give you an idea of the number of points you are likely to receive. Please note that the points indicated when you run these tests locally are not your final grade. Your solution will be graded by the TAs after you submit. Please also note that these test cases are not likely to be exhaustive and find every possible error. Part of programming is learning to test and debug your own code, so if something goes wrong, we can help guide you in the debugging process but we will not debug your code for you.

Project Overview.

In this Project, you will implement a class representing a Programming Competition with several different methods, and you will do so by using some structures we have covered (and possibly implemented) previously in the course.

In this programming competition, contestants can submit solutions to various problems (the number is unspecified) and receive a score, but only their *M* best scores count toward their total score. *M* is specified when the Competition starts (i.e. when the Competition object is instantiated). There is also no specific limit to the number of contestants.

You should implement your solution in a file called `Competition.java.`

You can get an idea of what is happening by looking at the input and output files.

Input Files

These files contain lists of names and scores. These scores represent scores for a single programming problem. A contestant's total score is determined by summing up their top *M* scores.

Output Files

These are the outputs for testing the various methods. For example, `totalScore_output1.txt` contains the output for testing the `totalScore` function given `input1.txt`. For that test, M is set to 3. You can see these specific values listed at the beginning of `CompTest.java`.

Required Methods

For testing purposes, these methods should return Strings. However, you will most likely want to calculate these in other methods first using other structures.

Before getting to the description of each method, please note the following guidelines regarding String format:

- For a String representation of a `Contestant`, use the `toString` method that is already provided.
- For a String representation of an array, you can use `Arrays.toString`.
- You may also use the `Arrays.sort` method for sorting an array when required.

Method Descriptions for `Competition.java`

Method Signature	Description
Competition(int M)	constructor that specifies that each contestant’s top M scores will be counted
void processScore(String name, int score)	processes an incoming score for the given name
String totalScore(String name)	return the given person’s current total score as a String; remember that total score is the sum of the person’s top M scores
String scores(String name)	return a sorted list of the person’s current top M scores as a String; you can see the String format for this in the output files–this format was what I got when I used Arrays.toString
public String top(int m)	return a String representation of a list of the top m Contestants (based on their total scores); note that this list may end up with more than m scores, but it should not end up with less; this is because if there are ties, you should include everyone in that list; keep adding until you reach or exceed m
public String bottom(int m)	return a String representation of a list of the bottom m Contestants (based on their total scores); note that this list may end up with more than m scores, but it should not end up with less; this is because if there are ties, you should include everyone in that list; keep adding until you reach or exceed m
public String rank(String name)	return <name>’s rank as a String; note that all Contestants with

	`the highest score are ranked as 1 and the rankings go from there; all Contestants with the same score should have the same rank`
`public String ranked(int rank)`	`return a String representation of a list of all the Contestants with the given rank; again, this can be done with Arrays.toString`

Provided Classes and Allowed Imports

I have provided the following classes that you may want to alter and use for your solution:

- `Hashtable.java` (with associated `Pair.java`)
- `MinPQ.java` (with associated `EmptyQueueException.java`)
- `Contestant.java` (from Project 1, which you will probably need to change)

The first two are my solutions to Project 5, and they work as expected with the expected runtimes for a hashtable with linear probing and a heap-based PQ. You are welcome to use them (and change them), but you may also ignore them if you prefer using your own implementations.

Additionally, you may import and use the following classes:

- `java.util.ArrayList` (but please use this wisely, considering the runtimes of the various operations)
- `java.util.Arrays` (particularly useful for turning arrays to Strings and for sorting arrays)
- `java.util.TreeMap` (which is an implementation of a red-black tree)

You may also use any other classes you implemented for previous assignments in this class, or you may implement new classes.

Runtime and Space Considerations

The purpose of this assignment is for you to consider the best data structures to use for a specific application. You should NOT use more space than is necessary for a particular purpose, and you should not implement things in unnecessarily inefficient ways.

For example, since only the top M scores count for a particular Contestant, you really should not be storing more than that for each Contestant. Additionally, I recommend that

you do not iterate through all the scores for a Contestant each time a new score comes in. That is unnecessary.
As a (possibly useful) hint, my solution uses all three of the structures we implemented in Project 5 (although I used the TreeMap instead of my red-black tree because it has more functionality than mine).

Submission

To submit your code, upload the following files to **lectura** and use the **turnin** command below. Once you log in to lectura and transfer your files, you can submit using the following command:

```
turnin csc345p6 Competition.java <otherfiles>
```

Upon successful submission, you will see this message:

```
Turning in:
     Competition.java-- ok
     <otherfiles>-- ok
All done.
```

Note: When I say `<otherfiles>`, I mean that you should submit all the files your solution needs.

Testing your code on lectura.
It is always a good idea to compile and run your code on lectura before you submit. In order to do that, you need to transfer all files to lectura that are necessary for testing (but only *submit* the ones that are required for submission.) Once the files are submitted, you can use the following commands in the terminal as needed.

`javac <name of file to be compiled>` – this will compile the given file
`javac *.java` – this will compile all .java files in the current folder
`java <name of file to be run>` – this will run the compiled file

Grading

Test	Points
1	13
2	14
3	17
4	16

Possible Deductions (besides late submissions):

Violation	Max Deduction
Not following directions	60 points
Illegal import	60 points
Wasted space	20 points
Inefficient Runtime	30 points
Bad Coding Style	5 points