Project 2

Course Statistics

Due by 10:00am on Friday, May 6, 2022.

What to do

(a) Solve the problems described below.
(b) Follow the submission instructions to prepare an electronic version of the assignment. Remember, if you do not follow the submission instructions, your assignment will not be graded and you will receive a grade of 0. Specifically, your assignment will not be graded if you submit individual source code files instead of a `JAR` file. See below for details.
(c) Submit your assignment as described below.

How to submit

In this course, you will submit an electronic version of your assignment on Moodle before class on the due date. Remember to use the provided cover page.

What to submit

(a) Cover page (you can download it here). You can also find it on Moodle under "Homework Assignments and Projects".
(b) The output from the final version of the class `CourseStatisticsTester`.
(c) The final version of `CourseStatisticsTester.java`.
(d) The final version of `CourseStatistics.java`.
(e) The final version of `Course.java`.
(f) The final version of `CourseComparator.java`.

Electronic version: You should submit *all* files needed to compile and run your project. Do not submit unnecessary files. Do not submit `*.class` files. All source files should be archived into a single `JAR` file named `lastname-project2.jar`. You can use the following command to create a `JAR` file named `lastname-project2.jar` consisting of `Class1.java`, `Class2.java`, and `Class3.java` files. Note that you need to list all files you want archived.

```
$ jar cvf lastname-project2.jar Class1.java Class2.java Class3.java
```

Submit output file as a text file (`*.txt`). All other files must be submitted as PDF files. You can submit multiple PDF files if needed.

PDF files can be prepared using `pdflatex`, `MS Word`, or other common word-processing tools. You can also "print" to PDF using a PDF printer. It does *not* work to take a plain text file and simply change the file name extension to `.pdf`. Please submit all your files using the Moodle *Assignments* tool.

Part 1: Project Overview

In this project, you will design and implement a simple program to calculate *course statistics* based on a text file containing course information. Your program will read in a text file and use a `Map` implemented using a hash table to store course information such that for any given course you will store its name, title, the total number of students who have taken the course, and the total number of student who have recommended the course. Further, you will calculate and use a recommendation score for a course: the percentage of all students who recommended the course. Because the file may contain multiple entries for a specific course, you will need to ensure that you maintain the overall totals for all courses. That is, if a course appears multiple times in the text file, you should have a single entry for that course but the associated totals should be a sum of the numbers provided in each entry.

Your program will support:

- Looking up a course description by using the course name.
- Choosing the best course to take given several options.
- Displaying a list of courses and their full descriptions.
- Displaying a list of courses sorted by their overall recommendation rate.

For this assignment, you will only be provided the input text file. This means that you **must** rely on the functionalities provided by Java to complete your project. You **may not** use the interfaces and implementations provided by the textbook.

Part 2: Input file and Reading from a file

Input file You are provided a file `courses.txt` containing fictitious course information as follows. The first line is the course name. The second line is the course title. The third line is the number of students who have taken the course. The fourth line is the number of students who have recommended the course, assuming a scheme where each student casts either a "yes" (+1) vote or a "no" vote (0). In the example below, 96 students took CPSC385 and 80 of them recommended it. That is, 16 of them did not recommend it. Sample entry:

```
CPSC385
Computer Security
96
80
```

The file `courses.txt` contains multiple entries for the same course. For example, another entry for CPSC385 is:

```
CPSC385
Computer Security
20
19
```

You can assume that the file is correctly formatted. That is, each course is described in exactly four lines and the number of students who recommended the course is a positive number less or equal to the number of student who took it. There is also a test course added exactly 5 times with the same values.

```
TEST000
Test course
1
1
```

Reading from a file Reading from a file in Java is straightforward. You will provide the file name as a command-line argument. That is, you will run your program using the following line.

```
java CourseStatisticsTester courses.txt
```

You can use the following code to read in the file one line at a time. This sample code reads in a line and prints it out.

```
import java.io.*;

class Tester {
  public static void main(String[] args) {
    if(args.length != 1) {
      System.out.println("Error: You need to provide a file name.");
      System.exit(1);
     }

    try {
      FileReader inFile = new FileReader(args[0]);
      BufferedReader in = new BufferedReader(inFile);
      String line;
      while((line = in.readLine()) != null)
        System.out.println(line);
    } catch(FileNotFoundException e) {
        System.out.println("Error: File " + args[0] + " not found.");
        System.exit(1);
    } catch(IOException e) {
      System.out.println(e);
      System.exit(1);
    }
 }
}
```

Part 3: Programming Requirements and Class Specifications

You should write these four classes according to the following specifications.

Class `Course`

- This class represents an individual `Course` object.
- Instance variables:
  - `String name` to store course name (e.g., CPSC215).
  - `String title` to store course title (e.g., Data Structures and Algorithms).
  - `int students` to store the total number of students who have taken the course.
  - `int votes` to store the total number of positive recommendation votes.
- Default as well as a parametrized constructor to set the instance variables.
- Necessary setters and getters for instance variables.
- `double getScore()` to calculate and return the recommendation score, the percentage of students who recommend the course.
- `String description()` to return a String representation of a full course description. See Part 7 for sample output.
- `String toString()` to return a String representation consisting of the course name and the recommendation score. See Part 7 for sample output.
- You may add `private` helper methods as needed.

Class `CourseStatistics`

- This class maintains the course statistics.
- It uses a `Map` implemented with a hash table for storage.
- It uses a `PriorityQueue` for recommending the best course to take. This queue uses a custom `Comparator` object (`CourseComparator`) to compare courses.
- It keeps track of the total number of courses added.
- It has a default constructor to initialize the instance variables as needed.
- `void addCourse(Course c)` to add a course `c` if it not yet stored or to update its totals it if is.
- `Course getCourse(String name)` to return a stored course given its name (e.g., CPSC215).
- `void findBest(List<Course> courses)` to find and display the best course to take based on its recommendation score. This method uses a `PriorityQueue` to pick one of the courses provided in the list `courses`.
- `void displayStats()` to display all course names and their recommendation scores sorted by the scores. Please see Part 7 for sample output.
- `void displayAll()` to display all courses and their descriptions provided by `description()`. Please see Part 7 for sample output.
- You may add `private` helper methods as needed.

Class `CourseStatisticsTester`

- This is a tester class for `CourseStatistics`.
- This class should produce the output as listed in Part 7.
- Notice that a user can look up and compare any number of courses.
- You may add `private` helper methods as needed.

Class `CourseComparator`

- This class implements `compare()` to compare the objects based on their recommendation score as provided by `getScore()`.

Programming Requirements

- Your program should implement the classes described above.
- Your program should produce the same output as listed in Part 7 given an input file `courses.txt`.
- You must use a hash table implementation of a `Map` and a `PriorityQueue`.
- Your program must not directly make use of any arrays or linked lists.
- Your program **must only** rely on the functionalities and building blocks provided by Java. That is, you cannot use any interfaces or implementation provided by the textbook or others.
- You must identify and add appropriate `import` statements.

There is a single file provided for this assignment: `courses.txt`. You must create all other files from scratch. The following files must be sufficient to compile and run your program: `courses.txt`, `CourseStatisticsTester.java`, `CourseStatistics.java`, `Course.java`, and `CourseComparator.java`.

Part 4: Testing

Your `CourseStatisticsTester` must produce the output from Part 7.

Part 5: Documentation

Source code documentation: Document your code using Javadoc tags as before. Specifically, make sure to add `@author`, `@version`, `@param`, `@return` and `@throws` tags as appropriate. Additionally, analyze and add a line to the header of each method stating its time complexity. For example:

```
* Time Complexity: O(n)
```

Add in-line comments to explain how your code works. If you are unable to complete the assignment or your program does not work properly, make sure to document every part of your code that causes issues. For more details about Javadoc, see pp. 51, 52 of our main textbook or the Javadoc Tool Home Page.

External documentation: Document your code using the provided cover page. You should document all arbitrary design and implementation choices that you made that are relevant to grading your project. As before, you should document any issues with your code.

Part 6: Grading Criteria

This project is much more structured than Project 1 and your implementation must follow the above requirements. Therefore, your grade will be based on the following items:

- **Correctness**: your program must properly perform the operations described in Part 1.
- **Design**: your program must be well designed, that is, it should properly utilize and implement object-oriented principles.
- **Implementation**: your program must adhere to the implementation requirements described in Part 3.
- **Testing**: your `main()` method should produce output included in Part 1.
- **Documentation**: your code should be properly documented using Javadoc, in-line comments and the cover page. Do not forget to add a complexity statement for each method.

Part 7: Sample output

```
*** Displaying courses and their full descriptions...
Displaying 19 courses.
CPSC333: Computer Networks has been taken by 58 students and has a 93% recommendation score.
CPSC399: Independent Study has been taken by 119 students and has a 94% recommendation score.
CPSC320: Analysis of Algorithms has been taken by 88 students and has a 77% recommendation score.
CPSC375: High-Performance Computing has been taken by 45 students and has a 91% recommendation score.
CPSC310: Software Design has been taken by 152 students and has a 88% recommendation score.
CPSC340: Principles of Software Engineering has been taken by 56 students and has a 85% recommendation score.
CPSC110: Visual Computing has been taken by 126 students and has a 80% recommendation score.
CPSC275: Introduction to Computer Systems has been taken by 80 students and has a 87% recommendation score.
CPSC352: Artificial Intelligence has been taken by 76 students and has a 68% recommendation score.
CPSC385: Computer Security has been taken by 116 students and has a 85% recommendation score.
CPSC219: Theory of Computation has been taken by 116 students and has a 78% recommendation score.
CPSC304: Computer Graphics has been taken by 98 students and has a 82% recommendation score.
CPSC315: Systems Software has been taken by 84 students and has a 88% recommendation score.
CPSC316: Foundations of Programming Languages has been taken by 56 students and has a 82% recommendation score.
CPSC115: Introduction to Computing has been taken by 134 students and has a 89% recommendation score.
CPSC203: Mathematical Foundations of Computing has been taken by 96 students and has a 87% recommendation score.
CPSC225: Event Driven Programming has been taken by 117 students and has a 76% recommendation score.
CPSC215: Data Structures and Algorithms has been taken by 67 students and has a 80% recommendation score.
TEST000: Test course has been taken by 5 students and has a 100% recommendation score.

*** Displaying courses sorted by their recommendation rates...
TEST000: 100%
CPSC399: 94%
CPSC333: 93%
CPSC375: 91%
CPSC115: 89%
CPSC310: 88%
CPSC315: 88%
CPSC275: 87%
CPSC203: 87%
CPSC385: 85%
CPSC340: 85%
CPSC304: 82%
CPSC316: 82%
CPSC110: 80%
```

```
CPSC215: 80%
CPSC219: 78%
CPSC320: 77%
CPSC225: 76%
CPSC352: 68%

*** Looking up courses...
Enter a course ID to look up statistics. Enter 's' to stop.
Course ID: CPSC215
CPSC215: Data Structures and Algorithms has been taken by 67 students and has a 80% recommendation score.
Course ID: CPSC333
CPSC333: Computer Networks has been taken by 58 students and has a 93% recommendation score.
Course ID: XYZ
No such course! Try again.
Course ID: s

*** Comparing courses...
Enter course IDs to compare. Enter 's' to stop.
Course ID: CPSC340
Course ID: CPSC275
Course ID: CPSC375
Course ID: XYZ
No such course! Try again.
Course ID: s
Best course to take: CPSC375: High-Performance Computing has been taken by 45 students and
has a 91% recommendation score.
```