Course:	**INFO1214 fall 2019 Project**
Professor:	**Bill Pulling (S1&2) ● Tony Haworth (S3) ● Mike Clarkson (S4)**
Project:	***Programming Contest Scoreboard***
Due Date:	**Code in FOL drop box by Friday, Nov. 29, at 2000 hrs (8 pm)**

Description:

Students from Fanshawe College have competed in the International Collegiate Programming Contest (ICPC) almost every year since 2002. While this competition draws teams from around the world, Fanshawe College competes directly in a regional face-off that includes universities and colleges from Ontario, Michigan, Illinois, Pennsylvania and Ohio. This regional contest is known as the ECNA (East Central North America) regional programming contest. In recent years our teams have travelled to Windsor, Ontario to participate. This year on October 26th there were 24 teams assembled in Windsor. However, there were actually 115 teams in total since the U.S. teams congregated at three other ECNA-designated sites that are in the United States.

The students from each school work in teams to solve a set of problems. This year there were eleven problems. Each problem could be solved by writing a console program in either Java, C/C++ or Python. Each team submitted a solution to a problem in the form of a source code file. The solutions were submitted to the judge using a Web-based application known as Kattis. The judge tested each submission for correctness and replied to the submitting team with either a "yes" meaning correct or a "no" meaning incorrect. When a team submitted an incorrect solution, they could attempt to correct their solution and resubmit. The winning team was the one that solved the most problems. However, in some cases there were multiple teams that solved the same number of problems. To break any ties or draws, the teams were then also ranked by the total accumulated time they used to submit their correct solutions. An additional penalty of 20 minutes was then added for each incorrect solution submitted along the way. For example, two teams solved six problems each, but "UofT Deep Blue" with 901 accumulated minutes was ranked higher than "McMaster Zetta" with 1,390 minutes. This ranking system is also explained here on the ECNA website under the heading "Scoring".

Throughout the five hours (300 minutes) of the contest, Kattis generated a log of all submissions made by every team along with the "yes" or "no" judgement for each submission. This log was used to generate a scoreboard summarizing the results for each team. Here is a partial scoreboard from this year:

```
ECNA Contest 2019 (Windsor Only)

Team             Slv/Time    P1    P2    P3    P4    P5    P6    P7    P8    P9   P10   P11

Brock Badgers       4/691   Y/4   N/0   N/0   N/0   Y/4   Y/1   Y/1   N/0   N/0   N/0   N/0
Brock Generals      1/112   N/0   N/0   N/0   N/0   N/0   Y/1   N/1   N/0   N/0   N/0   N/0
Fanshawe Black      2/176   N/3   N/0   N/0   N/0  N/19   Y/1   Y/1   N/0   N/0   N/0   N/0
Fanshawe Red        2/270   N/2   N/0   N/0   N/0   N/0   Y/2   Y/1   N/0   N/0   N/0   N/0
McMaster Giga       3/542   N/0   Y/2   N/0   N/0   N/0   Y/1   Y/1   N/3   N/0   N/0   N/0
```

Each row in the scoreboard represents a specific team. The columns contain the following information, from left to right:

- Team - the name of the team
- Slv/Time - the total number of problems solved / the total time used to solve the problems
- P*n* – for problem *n*, Y (solved) or N (unsolved) / the number of attempts submitted to the judge

For this project you will code a Java program that obtains data for each submission from the log file then generates and displays a contest scoreboard including a title and column labels (shown above). The output will also report the number of submissions, which is not seen above because only the top few rows of the scoreboard are shown. However, the full scoreboard is shown on page 4.

Basic requirements:

Create a main class called *Your_Initials_Scoreboard* containing your program's main method, but your replace *Your_Initials* with your initials. Also create a helper class called *Your_Initials_ScoreboardMethods* to contain any helper methods you code for your solution. All helper methods you create should go in this second class file. Note that the section "Use of Methods" later in this document details some methods that you must include in your solution. Make sure to read this before you begin writing your code!

The input files

Note that your program will not require any keyboard inputs from the user. Instead, your program will obtain input data from two tab-delimited text files as follows:

submissions.txt	Contains the data for all solutions submitted during the contest as follows: • The first line of the file contains a title or description of the contest which should be displayed at the top of your scoreboard. This can include spaces. • The second line contains the following four fields which are positive integers that can be stored using the *int* data type: 1. The number of teams 2. The number of problems in the problem set 3. The duration of the contest in minutes 4. The number of minutes to be added to a team's total time for each incorrect submission • Each remaining line in the file represents one team submission and contains the following four fields: 1. The id (number) of the team that made the submission, from 1 to the number of teams 2. The time (number of minutes from the start of the contest) when the submission was made, from 1 to the contest duration 3. The id (number) of the problem, from 1 to the number of problems 4. An uppercase letter, either 'Y' (meaning solved) or 'N' (meaning not solved) The first three fields are positive integers that can be stored using the *int* data type.
teams.txt	Contains data for each team entered in the contest. Each row represents one team with the following two fields: 1. A positive integer (an int) that is a team id (number), from 1 to the number of teams 2. A team name as a string which may contain spaces and could be any length

Reading the data

You will need to read the data from the two files (*submissions.txt* and *teams.txt*) into one or more data structures in your program to be processed. You are required to validate each line of input in the submissions file, which contains information about a single submission to the judge. You will need to verify that the team ID, the time of the submission, the problem ID, and the judgement all have reasonable values. You can use the information in the above section "The Input Files" to determine what values are valid for these fields. You should do this first before you use the data in your program. If any field of a submission is invalid you should not process it. However, you should keep track of how many valid submissions and how many invalid submissions were read by your program. This information will appear in your program's output later.

You may organize the data read-in from the files in your own way, however you will need to use a data structure to store the scoreboard data and this could involve some combination of arrays. One approach is to use three arrays to store all the data required to build the scoreboard. This approach is outlined in the following paragraphs.

A *submissions* array would hold the number of submissions (an integer) for each problem and for each team. This would be a two-dimensional array with one row for each team and one column for each problem. The second line of the *submissions* file contains the information you need to determine the size of this array. The row and column indexes would correspond to the team id and the problem id respectively. You would be able to populate this array by reading each line of the *submissions* file, field-by-field, using a loop and then incrementing the element in the *submissions* array at the corresponding row and column. Keep in mind that the team ids and the problem ids start at 1 while the row and column indexes start at 0.

A two-dimensional *times* array would be needed to store the time (an integer) for each problem solved by each team. Again, there would be one row for each team and one column for each problem so it would have the same dimensions as the *submissions* array. This array would also be populated while reading through the data in the *submissions* file. In fact, in could be done using the same file-reading loop! The array would only need to be updated if the last field of a line in the *submissions* file contains a 'Y' indicating that the problem was correctly solved with that submission. This would involve assigning the time of the submission to the array element at the corresponding row (team id) and column (problem id). Note that the time field for any submission that was <u>incorrect</u> (last field is 'N') does not have any effect on a team's total time.

The *teams* array would be a regular one-dimensional array of Strings with one element per team. This would be populated by reading the *teams* file and assigning one team name to each element of the array. The team id which is the first field on each line of the file would be used to determine which element of the array to assign the name to. In other words, the element index where a team name is stored should be related to the team id.

Printing the scoreboard from the data

For the basic requirements you do <u>not</u> need to rank the teams or sort them.

Assuming you are using the three arrays described in the previous section, and these arrays have already been populated by reading the *submissions* and *teams* files, then the steps to generate the scoreboard are as follows:

First you would print the scoreboard title and a row of column labels.

Next you would use a counter-controlled loop to iterate through all the teams by team id. In each iteration of this loop you would:

1. look-up and print the name of a team from the teams array using the team id to generate an array index. This goes in the first column of the scoreboard.
2. look-up and print the number of problems solved and the total time for the team, delimited by a '/'.
    a. To get the number of problems solved, look-up the corresponding row in the times array and use a loop to count the number of problems for which the time is not 0. Remember that the times of incorrect submissions are not stored in this array so those problems should have a time of 0 in the array.
    b. To get the total time for the team use a loop to accumulate (add) all the times in this same row of the teams array. However, you must add to this total time the time penalty assigned for each incorrect submission. You will have read this penalty time from the second row of the submissions file, multiply this by the number of incorrect submission and add it to the team's total time. HOWEVER, note that there should be no time penalty added for problems that were not ultimately solved by the team. Refer to the next page for a sample calculation.

Here’s an example of how the team’s total time is calculated:

```
column indexes->   0    1    2    3    4    5    6    7    8    9   10
                 ┌────┬────┬────┬────┬────┬────┬────┬────┬────┬────┬────┐  base time
         times-> │ 41 │ 0  │ 0  │108 │ 24 │ 0  │ 0  │ 0  │ 0  │235 │ 0  │  = 408
                 └────┴────┴────┴────┴────┴────┴────┴────┴────┴────┴────┘


column indexes->   0    1    2    3    4    5    6    7    8    9   10
                 ┌────┬────┬────┬────┬────┬────┬────┬────┬────┬────┬────┐  If penalty is 20
 # submissions-> │ 2  │ 7  │ 0  │ 1  │ 1  │ 0  │ 2  │ 0  │ 0  │ 4  │ 0  │  for a wrong answer
                 └────┴────┴────┴────┴────┴────┴────┴────┴────┴────┴────┘

                                                                           additional time
  # of penalized                                                           = 4 x 20
  submissions ->   1    0    0    0    0    0    0    0    0    3    0     = 80

                                                                           = 408 + 80
                                                      team’s total time    = 488 minutes
```

3. for each problem, look-up and print ‘Y’ if the problem was solved or ‘N’ if it was not, and the number of submissions made by the team for the problem, delimited by a ‘/’.

The scoreboard must be formatted so that each column of the table lines-up properly and there are at least two spaces between data in adjacent columns. You may truncate the team names (16 chars?) to limit the width of that column.

Lastly, after printing the scoreboard you should print information showing the number of valid submissions that were processed. If there were any invalid submissions you should also print this count.

The output generated by your program should look extremely similar to this:

```
ECNA Contest 2019 (Windsor Only)

Team              Slv/Time    P1    P2    P3    P4    P5    P6    P7    P8    P9   P10   P11

Brock Badgers        4/691   Y/4   N/0   N/0   N/0   Y/4   Y/1   Y/1   N/0   N/0   N/0   N/0
Brock Generals       1/112   N/0   N/0   N/0   N/0   N/0   Y/1   N/1   N/0   N/0   N/0   N/0
Fanshawe Black       2/176   N/3   N/0   N/0   N/0  N/19   Y/1   Y/1   N/0   N/0   N/0   N/0
Fanshawe Red         2/270   N/2   N/0   N/0   N/0   N/0   Y/2   Y/1   N/0   N/0   N/0   N/0
McMaster Giga        3/542   N/0   Y/2   N/0   N/0   N/0   Y/1   Y/1   N/3   N/0   N/0   N/0
McMaster Peta        2/414   Y/3   N/0   N/0   N/0   N/0   Y/1   N/4   N/4   N/0   N/0   N/0
McMaster Zetta      6/1390   Y/9   N/0   Y/5   N/0   Y/3   Y/2   Y/7   Y/4   N/0   N/0   N/0
Ryerson              3/464   N/0   N/0   N/0   N/0   N/2   Y/1   Y/1   Y/3   N/0   N/1   N/0
UofT Deep Blue       6/901   Y/2   Y/1   N/0   N/0   Y/3   Y/3   Y/4   Y/3   N/0   N/0   N/0
UofT Midnight B      4/907   Y/8   N/0   N/0   N/0   Y/2   Y/2   Y/1   N/2   N/0   N/0   N/0
UofT Royal Blue      7/628   Y/2   Y/1   N/3   N/0   Y/2   Y/1   Y/1   Y/2   N/0   N/0   Y/1
UTSC-A              5/1019   Y/4   N/0   N/0   N/0   Y/2   Y/2   Y/2   Y/4   N/0   N/0   N/0
UTSC-B              5/1061   Y/1   N/0   N/0   N/0   Y/3   Y/1   Y/1  Y/11   N/0   N/0   N/0
Waterloo Black     10/1172   Y/4   Y/1   Y/2  N/11   Y/2   Y/1   Y/1   Y/1   Y/1   Y/2   Y/1
Waterloo Gold      11/1394   Y/3   Y/2   Y/1   Y/1   Y/2   Y/1   Y/1   Y/1   Y/1   Y/1   Y/1
Waterloo Red        10/950   Y/1   Y/4   Y/1   Y/3   Y/1   Y/1   Y/1   Y/2   N/0   Y/1   Y/1
Waterloo White       8/903   Y/2   Y/2   Y/1   N/1   Y/2   Y/1   Y/1   Y/3   N/1   N/4   Y/1
Windsor Black        4/721   Y/2   N/0   N/0   N/0   Y/3   Y/3   Y/2   N/7   N/0   N/0   N/0
Windsor Blue           0/0   N/0   N/0   N/0   N/0   N/0   N/0   N/0   N/0   N/0   N/0   N/0
Windsor Green        1/243   N/1   N/0   N/0   N/0   N/0   Y/6   N/1   N/2   N/0   N/0   N/0
Windsor White        1/121   N/0   N/0   N/0   N/0   N/0   N/1   Y/1   N/0   N/0   N/0   N/0
Windsor Yellow       1/109   N/1   N/0   N/0   N/0   N/1   Y/2   N/1   N/0   N/0   N/1   N/0
york-lion1           4/783   Y/5   N/0   N/0   N/0   Y/6   Y/5   Y/3   N/1   N/0   N/0   N/0
york-lion2           4/756   Y/5   N/0   N/0   N/0   Y/2   Y/3   Y/1   N/0   N/0   N/0   N/0

316 valid submission(s) were processed.
3 submission(s) were invalid and ignored.
```

Use of methods:

We want you to minimize the amount of code in your main() method, and do much of the work in various helper methods in your “methods” class. You should include additional methods of your choosing to achieve this, but you must also define and use the following required methods:

- A Boolean method to validate the data for a submission. The method should accept all the fields read-in from the submissions file for one submission as well as the maximum valid values for the numeric field (e.g. the maximum team ID is the number of teams in the contest). It should return false if ANY field is outside the valid range of values for that field. It should only return true if every field in the submission is valid.
- A method to calculate the number of problems solved by a team. If your solution uses the arrays described in the section *Reading the Data* then this method should accept one row of the times array. The method will count and return the number of problems solved (i.e. the number of elements in the row that contain a time that is greater than 0).
- A method to calculate the total time for a team. If your solution uses the arrays described in the section *Reading the Data* then this method should accept one row of the times array, one row of the submissions array and the time penalty for incorrect submissions as documented in the second line of the *submissions* file. The method will accumulate the times for each problem solved and then add the penalty amount for each incorrect submission. There is sample calculation shown on the previous page under part 2b of the “Printing the Scoreboard from the Data” section.

NOTES:

- Part of the grade assigned to your project will be for defining and using suitable custom methods to streamline your main() method.
- All methods should be in a separate helper class named ***Your_Initials_*****ScoreboardMethods**.java. Make sure that you include this file when you submit your project.

Tips:

1. Make sure to compare your project with the *Marking Key* at the end of this document to make sure you have addressed everything required, including elements of style and organization.
2. There is no excuse to hand-in code that doesn’t compile. Use Eclipse to identify and help you correct any syntax errors.
3. Do the pseudo-code for this program first and include it in a comment block at the top of your program, just below your program documentation header. You can then copy and paste the portions of the pseudo-code to use as comments into each section of your code, but leave the original pseudo-code intact at the top of your code.

   This pseudo-code block may be more than one page long if you feel it is necessary, but the most important thing is that it clearly outlines the steps you will follow as you write your program. Also, if your program does not compile, or it crashes during testing, marks will be deducted here.

   Build the program one method at a time. Build a method, get it working, and test it to make sure it does what it is supposed to do.
4. All naming conventions for variables, constants, and methods must be followed.

Optional bonus requirements:

For 4 additional marks, incorporate the following additional requirements:

1. The rows of your scoreboard should be sorted by increasing rank. The rank of a team is determined first by the number of problems solved. A team solving more problems is ranked higher. Where there is a tie (two or more teams have solved the same number of problems) the team's total time will be used as a secondary sort key. The team with the lower time will be ranked higher. The team ranked highest has a rank of 1, the team ranked second-highest has rank of 2, and so on.

2. Each team's rank should be included in an additional column at the left side of the scoreboard.

Here is a sample (the first few rows) of what your output should look like when the bonus requirements are completed:

```
ECNA Contest 2019 (Windsor Only)

Rank  Team             Slv/Time    P1    P2    P3    P4    P5    P6    P7    P8    P9   P10   P11

1     Waterloo Gold     11/1394   Y/3   Y/2   Y/1   Y/1   Y/2   Y/1   Y/1   Y/1   Y/1   Y/1   Y/1
2     Waterloo Red       10/950   Y/1   Y/4   Y/1   Y/3   Y/1   Y/1   Y/1   Y/2   N/0   Y/1   Y/1
3     Waterloo Black    10/1172   Y/4   Y/1   Y/2  N/11   Y/2   Y/1   Y/1   Y/1   Y/1   Y/2   Y/1
4     Waterloo White      8/903   Y/2   Y/2   Y/1   N/1   Y/2   Y/1   Y/1   Y/3   N/1   N/4   Y/1
5     UofT Royal Blue     7/628   Y/2   Y/1   N/3   N/0   Y/2   Y/1   Y/1   Y/2   N/0   N/0   Y/1
6     UofT Deep Blue      6/901   Y/2   Y/1   N/0   N/0   Y/3   Y/3   Y/4   Y/3   N/0   N/0   N/0
7     McMaster Zetta     6/1390   Y/9   N/0   Y/5   N/0   Y/3   Y/2   Y/7   Y/4   N/0   N/0   N/0
```

Submitting your work:

- Submit your Java SOURCE FILES to the Project dropbox in FOL by the deadline indicated on page 1 of this document.
- If you teacher prefers a zipped .java file, please zip it up.

Submit your project on time!

Code submissions to the FOL drop box must be made on time! Late projects will be subject to late penalties as follows:

- A penalty of 10% will be deducted if your submission is received < 24 hours late.
- An additional 10% will be deducted for each additional 24 hour period.
- Your project will receive a mark of 0 (zero) if it is more than 5 days late.

Submit your own work!

1. It is considered cheating to submit work done by another student or from another source as your own work. This is called plagiarism. Helping another student cheat by letting them copy your source code files is called abetting plagiarism. Both are considered to be academic offences.
2. YOU MUST WRITE YOUR OWN CODE!
   Students are encouraged to share ideas and to work together on practice exercises, and you can help someone else to fix a bug in their code, but any code or documentation prepared for your project must be done by you. Penalties for committing plagiarism, or helping another student plagiarize by sharing source code with them, include a zero grade and an academic offence being filed against the student or students involved. All submissions will be analyzed using plagiarism detection software especially designed to analyze Java code.

Marking Key:

	Marks Available	**Marks Assigned**
STYLE: Appropriate pseudo-code statements are done. Documentation header is complete. Program compiles without errors and runs without crashing. Naming conventions followed, variable, constant, and method names are all representative of what each does.	2	
ORGANIZATION: All methods except main() are defined in a separate helper class. Methods are used effectively to minimize the amount of low-level logic in main() and to capitalize on the reusability of methods.	2	
Basic Requirements		
FILE HANDLING: Successfully opens and reads all the data in each input file and incorporates exception handling for the FileNotFoundException. Also properly closes any Scanner objects used to read from the files.	2	
METHOD: Defines and uses a custom method to validate all the fields in each submission contained in the submissions file. Also counts the number of valid and invalid submissions in the file.	2	
METHOD: Defines and uses a custom method to count and return the number of problems solved by a team using the submissions and times data for a team.	2	
METHOD: Defines and uses a custom method to calculate and return the total accumulated time for a team to submit any correct solutions to problems using the submissions and times data for a team and including the appropriate penalty time for incorrect submissions.	2	
OUTPUT CORRECTNESS: All fields in the output are attributed to the correct team and field and are correct.	4	
OUTPUT FORMATTING: Prints to the console the appropriate title from the submissions file, a row of column labels and one row per team containing all required fields as indicated in the requirements. All corresponding column labels and column data are lined-up correctly with at least two spaces between columns. Also reports the number of valid submissions and, if any, the number of invalid submissions.	4	
TOTAL	20	
Bonus Requirements *Will only be considered if basic requirements have been done correctly.*		
BONUS: The teams are correctly sorted by rank and the team rankings are displayed in the first column of the scoreboard.	4	
TOTAL		