CSC 110 Assignment 7: Data Analysis

Due:

Friday Dec. 5, 8pm.

Learning Outcomes

When you have completed this assignment, you should understand:

- How to use a class to instantiate objects.
- How to use instance methods.
- How to sort and search for data in an array.
- How to create and use an array of objects.
- How to write output to a file.

Your Task

Your task in this assignment is to create **C02EmissionReport.java**, which reads in a data file that contains information on carbon dioxide emissions per country (see example input file named carbon.data). The program then prints a report on the highest and lowest total CO2 emission producers, the highest and lowest per person CO2 emission producers and the ranking of Canada in the list. The report will be printed to an output file named **CarbonReport.txt**.

We have provided you with the CO2Data class (see CO2Data.java). **Do not change the code in this file. However, you must**:

  - In your program (a separate file), use an array of C02Data objects to store the data that your code reads from the input file.
  - Add comments to the code in CO2Data.java.

Your program must:

- Ask the user to enter the name of the input file. Note that the marker may use a different data file from the sample provided. It will have exactly the same format (same columns and data types), but may contain more or fewer rows and/or data values from a different year.
    - Continue to ask for another input file name if the one entered is invalid.
- Implement two methods to sort the data in the array, as specified below. You may choose any sorting algorithm that you like.
    - Note: you will need to sort by total emissions and also by per-person road emissions in order to produce the report.
- Your program must include and use the following methods, with method signatures EXACTLY as given below (your code may include more methods):

```
public static CO2Data[] readData(String filename)
```

```
public static void sortByTotalEmissions(CO2Data[] arr)
public static void sortByCO2PerPerson(CO2Data[] arr)
```

Input File Format (example carbon.data)

The first line contains descriptive names for the columns. Your code will need to skip this line when reading values.

The second line contains the number of records / countries in the file.

Each remaining line contains the name of a country (with no spaces in the name) followed by all of the values for that country.

You may assume that Canada will always be in the data file and that rows will be complete.

Output File Format

Your output file (CarbonReport.txt) should have contents that match the following format (but of course the values will depend on the input file):

```
The country with the lowest total emissions is France
The country with the highest total emissions is USA
Canada is ranked 3 out of 10 lowest for total emissions

The country with the lowest per person road emissions is India
The country with the highest per person road emissions is USA
Canada is ranked 9 out of 10 lowest for per person road emissions
```

What the marker will look for

The marker will check that your code:

- Compiles & runs on the lab machines.
- Works correctly with an alternate input file containing more or fewer records and different data values.
- Correctly handles the invalid input file error by asking for another filename.
- Implements the expected behaviour and produces the output file **CarbonReport.txt** with the expected output.
- Correctly uses an array of CO2Data objects to store the data.
- Correctly implements at least one sorting algorithm.
- Is well-structured through the use of meaningful methods, and includes the methods given above.
- Is appropriately documented and matches the style guidelines. Appropriate comments are included in both C02Data.java and CarbonAnalysis.java.

Hand In

Submit **C02EmissionReport.java** plus your **commented CO2Data.java** file via the assignments link on connex.

More to Discover

- Searching and sorting algorithms have been studied by Computer Scientists for nearly as long as the discipline has existed. The challenge is to ensure an efficient solution. (http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html provide demos of more than 15 algorithms!)
- In addition to arrays, there are other techniques for storing large amounts of data, i.e., Linked Lists: http://www.mycstutorials.com/articles/data_structures/linkedlists
- These are topics that are considered in CSC 115, why not give it a try?