Joanna Klukowska
joannakl@cs.nyu.edu

Programming Project 1:
NYPD Motor Vehicle Collisions Analysis

Due date: Sept. 26, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating. If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will provide a tool for extraction of certain type of information about motor vehicle collisions in New York City. The New York Police Department (NYPD) provides data regarding all motor vehicle collisions that occur on streets on NYC. This data can be downloaded from NYC Open Data website at
`https://data.cityofnewyork.us/Public-Safety/NYPD-Motor-Vehicle-Collisions/h9gi-nx95`.
To simplify your task, the course website has a listing of several preprocessed files that contain the data from the last two years. Your program should run with one of those files as the input and extract some interesting information (details below) from it: we'll look for zip codes with most collisions, zip codes in which the collisions are deadliest, the areas that cyclists should avoid, etc.

The program that you write has to be command line based (no graphical interface) and it should not be interactive (do not ask the user for anything).

Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- working with multi-file programs
- reading data from input files
- writing data to output files
- using command line arguments
- working with large data sets
- using `ArrayList` class
- writing classes

Most, if not all, of the skills that you need to complete this project are based on the material covered in cs101. But there might be certain topics that your section did not cover at all, or that you did not get much practice on. Make sure to ask questions during recitations, in class and on Piazza.

Start early!

The Program Input and Output

You may use any Java classes for reading from input files and for writing to output files. But you need to understand how these classes work. In cs101, you should have learned how to use `Scanner` class for reading from input streams (input file is a type of stream), and how to use `PrintWriter` for writing to output files.

Joanna Klukowska
joannakl@cs.nyu.edu

Input File

Your program is given the name of the input text file as its command line argument (the first and only argument used by this program). The input file has the following structure:

- the first line contains column headings (your program can simply ignore it)
- all remaining lines contain 21 **comma separated entries** of data (your program may not need all of them, but they are all in the file)
  - date
  - time
  - borough
  - ZIP code
  - lattitude
  - longitude
  - on street name
  - cross street name
  - number of persons injured
  - number of persons killed
  - number of pedestrians injured
  - number of pedestrians killed
  - number of cyclists injured
  - number of cyclists killed
  - number of motorists injured
  - number of motorists killed
  - contributing factor vehicle 1
  - contributing factor vehicle 2
  - unique key
  - vehicle type code 1
  - vehicle type code 2

  Note that some of the entries may contain multiple words, for example, "WEST 53 STREET" is a street address. In general, all text entries are surrounded in double quotes. This is to allow commas within the entries (any comma within double quotes is not a field delimiter, but part of that field). Numerical data fields are not surrounded in double quotes.

If a line in the input file does not contain correct number of comma separated entries, it should be ignored. The program should continue to the next line. This means that the code that parses the input file needs to handle incomplete lines.

If the filename is omitted from the command line, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: missing name of the input file").

If the filename is given but the file does not exist or cannot be opened for reading by the program, for any reason, it is an error. The program should display an error message and terminate. The error message should indicate what went wrong (for example: "Error: file `collisions.csv` does not exist.", but make sure to replace the name with the name of the file with which the program was called).

Your program is NOT ALLOWED to hardcode the input filename in its own code. It is up to the user of the program to specify the name of the input file. Your program should not modify the name of the user-specified file (do not append anything to the name).

Your program may not read the input file more than once.

Your program may not modify the input file.

Joanna Klukowska
joannakl@cs.nyu.edu

Output File

Your program should produce an output text file. Its name should be constructed based on the input file name: it should match the input file name without the extension (if any extension is present in the input file name) and it should have `.out` extension. For the purpose of this project, you may assume that the file extension is any substring of the file name that follows the last dot (`.`) in the name. For example

- if the input file name is **`NYPD_Motor_Vehicle_Collisions_all_2015.csv`**, the output file should be named **`NYPD_Motor_Vehicle_Collisions_all_2015.out`**,
- if the input file name is **`NYPD_Motor_Vehicle_Collisions_Manhattan_2015_August_24`**, the output file should be named **`NYPD_Motor_Vehicle_Collisions_Manhattan_2015_August_24.out`**.

If a file with the given filename exists already, your program should overwrite it without any warning. If the program cannot create a file with the given name, for any reason, it should print an error message and terminate. The error message should indicate what went wrong (for example: "Error: cannot create file **`collisions.out`**.", but make sure to replace the name with the name of the file that the program is trying to actually create).

User Input

This program is not interactive. It should not prompt or expect any input from the user. In fact, your program should not assume that a human is running it.

Computational Task

Once the program reads the data from the input file, it should perform the following tasks. (The program can perform these tasks in whatever order you wish, but the output file should contain the data in the order and format specified below.)

Note1: the sample outputs are made up and do not correspond to any particular input file.

Note2: the input files may contain incorrect information - it is not the job of your program to detect that.

Note3: your data can be printed using upper case, lower case or mixed letters.

Task 1 Find the three zip codes with the largest number of collisions. The output should be formatted as follows.

```
ZIP codes with the largest number of collisions:
    10021    121 collisions
    10016     95 collisions
    11372     89 collisions
```

Task 2 Find the three zip codes with the smallest number of collisions (only zip codes listed in the input file should be used). The output should be formatted as follows,

```
ZIP codes with the fewest number of collisions:
    10021    13 collisions
    10016     5 collisions
    11372     2 collisions
```

If there is a tie in task 1 and/or task 2, all zip codes with the same count should be provided. For example

```
ZIP codes with the largest number of collisions:
    10021    121 collisions
    10023    121 collisions
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
    10016     95 collisions
    11372     89 collisions
    11103     89 collisions
```

Task 3 Find the three zip codes that have the most injuries and fatalities resulting from vehicle collisions (overall for all persons involved). The output should be formatted as follows.

```
ZIP codes with the most injuries and fatalities (combined):
    10021    13 injuries and fatalities
    10016     5 injuries and fatalities
    11372     2 injuries and fatalities
```

If there is a tie, it should be resolved based on the number of fatalities that occur in the zip codes involved in the tie. If both values are identical for two zip codes, they should both be printed to the output file (like the tie resolution for tasks 1 and 2).

Task 4 Find the three zip codes that are most dangerous for the cyclists (i.e., the number of cyclist injuries and fatalities is the largest). The output should be formatted as follows.

```
ZIP codes with the most cyclist injuries and fatalities:
    10021    13 cyclists hurt
    10016     5 cyclists hurt
    11372     2 cyclists hurt
```

If there is a tie, it should be resolved based on the number of cyclist fatalities that occur in the zip codes involved in the tie. If both values are identical for two zip codes, they should both be printed to the output file (like the tie resolution for tasks 1 and 2).

Task 5 Find the percentage of collisions involving the following vehicle types:

- taxi
- bus
- bicycle
- fire truck
- ambulance

The output should be formatted as follows:

```
Percentage of collisions involving certain vehicle types:
    taxi        15.03%
    bus          2.43%
    bicycle      5.12%
    fire truck   0.54%
    ambulance    0.89%
```

Notice that the numbers should be displayed with two decimal spaces and decimal points should be aligned.

Task 6 Find something that interests you and report on it.

Program Design

Your program must contain three classes:

- `Collision` class to represent a single collision. An object of class `Collision` contains all the information about a single collision.

Joanna Klukowska
joannakl@cs.nyu.edu

- `CollisionList` class to represent all the `Collision` objects in a single container. `CollisionList` class should store all the `Collision` objects in an `ArrayList` of collisions (depending on your design, you may use more than one such `ArrayList`). This class should provide methods that return the results needed by the five tasks described in the previous section.
- `CollisionInfo` class that is the runnable program containing the `main()` method. This class is responsible for parsing the command line argument, reading the input file, creating the `CollisionList` object based on the input file, calling all the methods of the `CollisionList` object needed for completion of the five tasks, and writing to the output file. This class should have methods other than the `main()` method.

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at http://cs.nyu.edu/~joannakl/cs102_f15/notes/CodeConventions.pdf.

You must document all your code using Javadoc. Your class documentation needs to provide a description of what it is used for and the name of its author. Your methods need to have description, specification of parameters, return values, exceptions thrown and any assumptions that they are making.

Class's data fields and methods should not be declared `static` unless they are to be shared by all instances of the class or provide access to such data.

You may use `ArrayList` and `String` classes and any methods that are provided in them. You may not use any of the hash table classes like `HashTable`, `HashMap`.

You may use any classes for reading the input from the file and writing the output to the file.

You may use any exception related classes (if you wish).

You may use the sort/search methods provided in the `Arrays` and/or `Collections` classes (for that, your `Collision` class should implement the `Comparable` interface).

Working on This Assignment

You should start right away! This program does not require you to write much code (well, more than you are used to from cs101, but less than future assignments), but it will take some time.

You should modularize your design so that you can test it regularly:

- Start with a program that reads in the input file. Make sure that this works. (This can be tested easily, by simply writing the data back to an output file or to the screen without any changes.)
- Write a class to represent a collision. Make sure that this works. Think about all methods that this class may need to provide. Test it with a small input file or just create a few `Collision` objects and test all of the methods.
- Write a class that represents the list of collisions. Again, test with with a small input file and/or `Collision` objects created directly in the code.
- Write the main() function that combines the above pieces and produces the output file.
- Test, test, test, test. Use different input files. Try to break your program (this is what the graders will be doing).
- Re-read this project specification and make sure that your program adheres to it.
- Submit your code.

You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due date - make sure that you have working code if that happens.

Joanna Klukowska
joannakl@cs.nyu.edu

Grading

Make sure that you are following all the rules in the **Programming Rules** section above.

If your program does not compile or crashes (almost) every time it is ran, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

15 points design, implementation and correctness of the `Collision` class,

40 points design, implementation and correctness of the `CollisionList` class (this includes the correctness of the results for the six tasks),

25 points design, implementation and correctness of the `CollisionInfo` class,

20 points proper documentation and program style.

How and What to Submit

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to NYU Classes. If you are unsure how to create a zip file or how to get to your source code files, you should ask long before the project is due.

If you wish to use your (one and only) freebie for this project (one week extension, no questions asked), then complete the form at `http://goo.gl/forms/YFDVBlscEB` **before the due date for the assignment**. All freebies are due seven days after the original due data and should be submitted to NYU Classes.