Joanna Klukowska
joannakl@cs.nyu.edu

Programming Project 5:
NYPD Motor Vehicle Collisions Analysis

Due date: Dec. 7, 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 for different time periods and locations. 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 detail reports for particular (user specified) zip codes and particular time frames.

The program that you write has to be command line based (no graphical interface).

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
- using command line arguments
- working with large data sets
- implementing an AVL tree
- writing complex code projects

Start early! This project requires significant code to be written.

The Program Input and Output

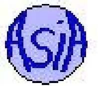
You may use any Java classes for reading from input files. But you need to understand how these classes work.

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). Note: this implies the user should not be prompted for the name of the input file by the program itself.

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

Joanna Klukowska
joannakl@cs.nyu.edu


`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.

The input file has the following structure:

- the first line contains column headings (your program can simply ignore it)
- all remaining lines contain up to 29 **comma separated entries** of data (your program may not need all of them, but they are all in the file) ; in some rows the might be fewer than 29 entries; there have to be at least 24 entries on the line (some of those entries may be blank) for the line to be valid; the following is the list of all the columns in the data set together with the corresponding type that should be user to represent it:
  - date (Date type, provided with the assignment)
  - time (String)
  - borough (String)
  - ZIP code (String)
  - lattitude (double)
  - longitude (double)
  - on street name (String)
  - cross street name (String)
  - number of persons injured (int)
  - number of persons killed (int)
  - number of pedestrians injured (int)
  - number of pedestrians killed (int)
  - number of cyclists injured (int)
  - number of cyclists killed (int)
  - number of motorists injured (int)
  - number of motorists killed (int)
  - contributing factor vehicle 1 name (String)
  - contributing factor vehicle 2 name (String)
  - contributing factor vehicle 3 name (String)
  - contributing factor vehicle 4 name (String)
  - contributing factor vehicle 4 name (String)
  - unique key name (String)
  - vehicle type code 1 name (String)
  - vehicle type code 2 name (String)
  - vehicle type code 3 name (String)
  - vehicle type code 4 name (String)
  - vehicle type code 5 name (String)

  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 (at least 24 entries should be present), 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.

User Input

The program should prompt the user for the zip code, start date and end date. The program should validate that the zip code is in a correct format. Valid zip codes have exactly 5 characters and all characters are digits. The program should validate that the dates are in a correct format. The valid dates should be written as mm/dd/yyyy (the month and day can be either one or two digits long, the year has to be four digits long). The start date should be smaller than (earlier) than the end date. After the program displays the results, the user should be prompted for the zip code, start date and end date again. The program should terminate when the user enters "quit" in place of the zip code.

Joanna Klukowska
joannakl@cs.nyu.edu

Computational Task

Once the user enters the information described in the **User Input** section, the program should display a report based on the data from the input file. The report contains summary of the collisions that occured in the given zip code within the specified dates (including both end dates). The report has to list the total number of fatalities and injuries together with the breakdown of each for pedestrians, cyclists and motorists.

The following two examples show sample user interaction and program output.

Output:

```
Enter a zip code ('quit' to exit): 10012
Enter start date (MM/DD/YYYY): 1/1/2017
Enter end date (MM/DD/YYYY): 11/20/2017

Motor Vehicle Collisions for zipcode 10012 (01/01/2017 - 11/20/2017)
====================================================================
Total number of collisions: 708
Number of fatalities: 0
         pedestrians: 0
            cyclists: 0
           motorists: 0
Number of injuries: 108
       pedestrians: 36
          cyclists: 30
         motorists: 45
Enter a zip code ('quit' to exit): quit
```

Output:

```
Enter a zip code ('quit' to exit): 113373
Invalid zip code. Try again.

Enter a zip code ('quit' to exit): 10013
Enter start date (MM/DD/YYYY): 01/01/17
Enter end date (MM/DD/YYYY): 11/20/17
Invalid date format. Try again.

Enter a zip code ('quit' to exit): 10013
Enter start date (MM/DD/YYYY): 7/1/2017
Enter end date (MM/DD/YYYY): 8/31/207
Invalid date format. Try again.

Enter a zip code ('quit' to exit): 10013
Enter start date (MM/DD/YYYY): 7/1/2017
Enter end date (MM/DD/YYYY): 8/31/2017

Motor Vehicle Collisions for zipcode 10013 (07/01/2017 - 08/31/2017)
====================================================================
Total number of collisions: 192
Number of fatalities: 1
         pedestrians: 1
            cyclists: 0
           motorists: 0
Number of injuries: 27
       pedestrians: 6
          cyclists: 6
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
         motorists: 15
Enter a zip code ('quit' to exit): 11103
Enter start date (MM/DD/YYYY): 9/1/2017
Enter end date (MM/DD/YYYY): 9/31/2017

Motor Vehicle Collisions for zipcode 11103 (09/01/2017 - 09/31/2017)
====================================================================
Total number of collisions: 29
Number of fatalities: 0
         pedestrians: 0
            cyclists: 0
           motorists: 0
Number of injuries: 7
       pedestrians: 0
          cyclists: 2
         motorists: 5
Enter a zip code ('quit' to exit): quit
```

Program Design

Your program must contain the following classes.

`Collision` class

An object of `Collision` class represents a single collision (single valid row from the input file).

This class should contain a one parameter constructor. The single parameter to this constructor should be an `ArrayList<String>` object that contains all the parsed entries from the input file. The signature for this constructor should be

```
public Collision (ArrayList<String> entries)throws IllegalArgumentException
```

If the line of the input file is the following:

```
03/13/2017,21:00,BROOKLYN,11223,40.59985,-73.9613,"(40.59985, -73.9613)",,,2337 CONEY ISLAND
 AVENUE ,0,0,0,0,0,0,0,0,Driver Inattention/Distraction,Unspecified,,,,3631843,SPORT UTILITY
 / STATION WAGON,PASSENGER VEHICLE,,,
```

then the corresponding `ArrayList` object should be

```
[03/13/2017, 21:00, BROOKLYN, 11223, 40.59985, -73.9613, (40.59985, -73.9613), , , 2337 CONEY
 ISLAND AVENUE , 0, 0, 0, 0, 0, 0, 0, 0, Driver Inattention/Distraction, Unspecified, , , ,
 3631843, SPORT UTILITY / STATION WAGON, PASSENGER VEHICLE, , ]
```

Note that all the values between the commas are strings and some of those strings are emtpy (i.e., `""`).

The constructor should validate that the following entries meet certain requirements:

- the date cannot be empty and has to represent a valid `Date` object (see below)
- the zip code has to be a five character string with digits as all of its characters
- the number of persons/pedestrians/cyclists/motorists injured/killed has to be a non-negative integer
- the unique key has to be a non-empty string

The other entries do not need to be verified and may contain empty strings.

The class should provide getters for each of the above values. The functions should follow the standard accessor naming conventions: `getZip()`, `getDate()`, `getKey()`, `getPersonsInjured()`, `getPedestriansInjured()`, `getCyclistInjured()`,

getMotoristsInjured(), getPersonsKilled(), getPedestriansKilled(), getCyclistKilled(), getMotoristsKil
().

The class should implement Comparable<Collision> interface. The comparison of two Collision objects should be based on the zip codes, dates and unique keys (in that order). This means that if two zip codes are different, then the comparison is based on the zip codes alone. If they are the same, then the comparison is based on the dates. If the date are the same, then the comparison is based on the unique keys.

The class should implement the equals() method to override the method that it inherits from the Object class. The criteria for comparison should be the same as for the Comparable interface.

CollisionsData class

An object of this class stores all of the collision records.

This class should be an AVL tree. You should use the provided BST tree implementation and specialize it to work with Collision objects class. The class should modify the add and remove methods so that the tree keeps the AVL tree structure. The class should add other private methods that are required for implementation of the AVL tree (for example, methods for updating the height, calculating balance factors and performing rotations). You should modify the internal node class to keep track of the heights of nodes.

This class should also provide methods that generate the report based on the user specified zip code and date range. Note: this class should not print to the standard output directly. Any reports that are generated should be returned in the form of a string, so that the calling class can use it in whatever way it needs to.

This class should implement the following public methods:

public void add(Collision item) adds the Collision object to the CollisionsData object

public boolean remove(Collision target) removes the Collision object from the CollisionData object; returns true if the target was removed, false if the target was not found and, hence, not removed

public String toStringTreeFormat() returns a string that contains graphical representation of the tree (this method is implemented in the BST_Recursive class and it should be included in the CollisionsData class as given

public String getReport (String zip, Date dateBegin, Date dateEnd) returns a string containing information about the numbers of fatalities and injuries for a given zip code and date range. For example, if the zip code is 11103 and the date range is 09/01/2017 - 09/31/2017, the returned string should be:

```
Motor Vehicle Collisions for zipcode 11103 (09/01/2017 - 09/31/2017)
====================================================================
Total number of collisions: 29
Number of fatalities: 0
         pedestrians: 0
            cyclists: 0
           motorists: 0
Number of injuries: 7
       pedestrians: 0
          cyclists: 2
         motorists: 5
```

CollisionInfo class

This class is the runnable program containing the main() method. This class is responsible for parsing the command line argument, reading the input file, creating the CollisionsData object based on the input file, interacting with the user, calling all the methods of the CollisionsData object needed for the output. This class may have methods other than the main() method to provide more modularity.

Restriction: All of the Collision objects have to be stored in a single CollisionsData object.

Joanna Klukowska
joannakl@cs.nyu.edu

Given Code

You should have recieved three pieces of code with this project.

BST_Recursive class provides the recursive implementation for a binary search tree. This class does not need to be included in the final project. You should use it to guide you through the implementation of the `CollisionsData` class that is an AVL tree. Feel free to copy/use any parts of this class that may be useful for the `CollisionsData` class. Keep in mind that the `BST_Recursive` class is a generic class. Your `CollisionsData` class is not.

Date class provides a simple representation for dates. Read through the code of this class to make sure you know how to use it.

splitCSVLine method allows you to parse the lines from the input file into an `ArrayList<String>` object containing all of the entries. This function produces empty strings to represent entries that were blank within the input line.

Efficient Design

The design of all of your classes has to take efficiency into consideration. This section provides a listing of some of the things that you should keep in mind. There may be other efficiency considerations not listed here.

The input file should be read only once. Reading data from disk is slow. The program needs to read the file only once and store the information in the memory of the program (this is the purpose of the `CollisionsData` class).

The tree used for storing the data should be an AVL tree. In theory, you could use the ordinary BST to store all the `Collision` objects (this, in fact, may be a good starting point for the project). The problem with using BST, though, is that it does not guarantee that the tree will be balanced. This may result in access times that take too much time.

The search for objects matching the given zip code must not visit all of the nodes in the AVL tree. In order to generate a report, the program needs to be able to locate all `Collision` objects within a particular zip code and date range. This could be done using a tree traversal. But this process is very inefficient since the objects matching the query are a small fraction of all of the objects stored in the tree. Given the natural ordering of the `Collision` objects, all the objects matching a particular zip code and date range must be located in a very specific area of the tree. Your task is to figure out the algorithm for retrieving those objects in the proportional to the number of matching objects.

Examples of efficient way of traversing the `CollisionData` object in search for matching `Collison` objects.
Consider the AVL tree in figure 1. In each node, assume that the letter indicates the zip code and the number indicates the date, and the two character string indicates the unique id.

An efficient implementation should visit only the nodes shaded in gray when searching for the `Collision` objects whose zip code matches ”F” and whose date range matches 5-40.

Similarly, an efficient implementation should visit only the nodes shaded in blue when searching for `Tree` objects whose name matches ”A” and whose date range matches 0-100.

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at https://joannakl.github.io/cs102_f17/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.

Joanna Klukowska
joannakl@cs.nyu.edu

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).

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.

50 points class correctness: correct behavior of methods of the required classes and correct behavior of the program

20 points design and the implementation of the required classes and any additional classes

10 points program efficiency

20 points proper documentation, program style and format of submission

How and What to Submit

For the purpose of grading, your project should be in the package called `project5`. This means that each of your submitted source code files should start with a line:
`package project5;`

Your should submit your source code files (the ones with .java extensions only) in a single **zip** file to Gradescope.

You can produce a zip file directly from Eclipse:

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next
- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select ”Save in zip format”, ”Compress the contents of the file” and ”Create only selected directories”
- click Finish

Joanna Klukowska
joannakl@cs.nyu.edu

Figure 1: Efficient search for matching species.