CSCI-605 Advanced Object-Oriented Programming Concepts
Homework 6

1 Introduction

You work for an organization that oversees the running of a toll-based highway. This toll system infrastructure has the following characteristics:

- Video cameras at all exits record each vehicle's arrival or departure at an exit on the highway.
- The images are converted into the alphanumeric values imprinted on the vehicles' license tags.
- An entry is added to a database that includes the license tag, exit number, and time. Note that there is no indication of whether the vehicle was arriving or departing. This is deduced by the timewise sequence of entries.

Your job is to process the database, generate some reports, and then answer some user queries about use of the highway based on that data.

1.1 Goals

The goal of this lab is to apply what you have learned about the Java Collections Framework to a practical problem, make sound choices about which data structure works best for any particular set of needs, and use them correctly in your programs. Note that "works best" includes a number of factors including how the data is stored and retrieved, as well as performance.

1.2 Provided Files

None of the provided files should be modified in your solution.

1. `FileHandler` - This class provides a static method that, when given a filename, will return an `Iterable<String>` of lines in the file. It is recommended that a `static` import is used (i.e. `import static hw6.FileHandler.*;`) so that the `open` method can be used in a `for` loop., e.g. `for(String line : open("afile.txt")) { ... }`. If the `FileHandler` encounters any problems (e.g. the file does not exist), it will print an error and the program will exit.

2. `Diff` - You should be familiar with this utility by now; use it to compare your output to the provided examples to insure that yours matches the expected output.
3. `TollsRUs` - An interface that provides a set of useful constants that you will want to use when implementing your solution.
4. `Exit` - A class that represents an exit on the highway including the exit number, name, and mile marker. This is a utility class that is used by the `ExitInfo` class.
5. `ExitInfo` - A provided class that contains only `static` methods for retrieving information about exits and tolls on the highway.
   (a) `isValid(int)` - Returns `true` if the given exit number is valid, and `false` otherwise.
   (b) `getName(int)` - Returns the official name of the exit with the specified number.
   (c) `getMileMarker(int)` - Returns the mile marker for the exit with the specified number as a `double` (i.e. its distance in miles from the highway's origin).
   (d) `getToll(int, int)` - Returns the dollar amount of the toll between the given arrival and departure exits.
6. `data` - This directory includes several data files that contain toll event information for vehicles traveling on the highway (see "Program Operation" for more detail). These files may be used when implementing and testing your program.
7. `output` - This directory contains examples of the expected output from the program when it is implemented and executed. You will find a detailed description of the file contents below.

1.3 Program Operation

The "database" is actually just a text file, each line of which includes the information regarding a single *toll event*. A toll event occurs when a vehicle arrives or departs at an exit (though the event does not indicate the difference). You will use the `FileHandler.open(String)` method to return the contents of the file as an `Iterable<String>`. You will be responsible for parsing each line in the file and adding/updating the information in your Toll Database.

When executed the program displays the following information.

1. The number of completed trips, i.e. vehicles that arrived on the highway at some exit, and later departed from the highway at some other exit.
2. A list of all the vehicles that are still on the highway, sorted by license tag, i.e. vehicles that arrived on the highway at some exit but have not yet departed.
3. A list of all cases of speeding, sorted by license tag, and then by time, i.e. vehicles that completed a trip in less time than is possible while traveling at or below the highway's maximum speed limit.
4. Billing for all complete trips, sorted by license tag, and then by time.

Then the program prompts the user for further instructions.

- Show the bill for a particular license tag. Each completed trip is shown in the same form it is shown in the full billing listing, and the trips are sorted by time. A total for that tag is displayed at the end.
- Show activity at a particular exit, sorted by time. This includes any vehicle that arrived on, or departed from this exit.

- Quit the program; the user may repeat any combination of the previous commands until they quit the program.

1.4 Implementation

While you are free to design and implement your program as you see fit (as long as it meets the requirements detailed above), below are some suggestions for classes that you should consider implementing.

- Toll Record - A *toll record* represents a single vehicle trip on the highway. A toll record is created when a vehicle arrives on the highway via an exit and includes the arrival exit number and arrival time. The record is completed at some later time when the vehicle departs the highway via some other exit. An incomplete record will include an arrival but no departure, indicating that the vehicle is still on the highway. A toll record class should implement the behavior described below. Note that the following list is not comprehensive.
  - A method to compute and return the toll for the vehicle. This should only be called if the record is complete (i.e. it contains a departure exit and time). The toll charged for each trip should be determined using the `ExitInfo.getToll(int, int)` method (see above).
  - An `equals(Object)` method. Two toll record objects are considered equal if all of their fields are equal.
  - A `toString` method that returns a string representation of the toll record in the format `[tag]`{`(arrival-exit,arrival-time)`} if the record is incomplete or `[tag]`{`(arrival-exit,arrival-time), (departure-exit,departure-time)`} if the record is complete.
  - A method that returns a *report* for the toll record as a string in the format
    `[tag] on` ***`on-exit`***`, time` ***`on-time`***`; off` ***`off-exit`***`, time` ***`off-time`***
  - A method that returns a hash code for the toll record. It should meet the requirements of the general contract for `equals` and `hashCode` (i.e. if `a.equals(b)` then `a.hashCode() == b.hashCode()`).
  - A toll record must define a *natural ordering* such that toll records are arranged in some order. You may choose the ordering based on the requirements.
- Toll Road Database - The *toll road database* keeps track of toll records. A Toll Road Database class should implement the behavior described below.
  - A constructor that accepts the name of a toll event data file as a parameter. The constructor should use the `FileHandler` class to read the file as an `Iterable<String>`. Each line in the file will be in the format `time,tag,exit`. These toll events should be used to populate the toll record database. Keep in mind that the lines in the file do not indicate whether the event is an arrival or a departure. Your implementation must determine this based on the order of events, i.e. the odd numbered events for a given tag indicate an arrival, and the even numbered events indicate a departure.
  - A method that returns a *summary of completed trips* as a string.

  - A method that returns an *on-road report* listing the vehicles that are still on the highway as a string. The vehicles will be ordered based on license plate tag.
  - A method that returns a *billing report* for all completed trips as a string. The report lists the trips first by vehicle tag and then by arrival time for the vehicle's trip.
  - A method that returns a *speeder report* listing vehicles that exceeded the maximum speed limit during a completed trip on the highway. The `TollsRUs.SPEED_LIMIT` constant should be used as the speed limit.
  - A method that returns a report for a single customer as a string. Customers are identified using the license tag for their vehicle. The report should include all toll records for completed trips in order by arrival time (incomplete trips will not be included because they cannot be billed) followed by a total amount due.
  - A method that returns activity at a specific exit. This will print all the toll records that include the exit as the arrival or departure point. Completed records are listed first, in order by vehicle tag, and then by the arrival time. Any incomplete trips are listed next, using the same ordering.
- Toll Report - The main, executable class. This class should define a `main` method that accepts a single command line argument specifying the name of a toll event data file, e.g. `java hw6.TollReport data/5guys.txt`. The class should perform the following actions:
  1. Create a new Toll Record Database with the specified toll event data file.
  2. Use the Toll Record Database to print a full report including a summary of completed trips, an on-road report, a speeder report, and a billing report.
  3. Provide a command line interface into which the user may enter the following commands:
     - `b` *`tag`* - prints the bill for a single license tag.
     - `e` *`exit`* - prints the activity report for the exit with the specified number.
     - `q` - quits the program.

1.5 Testing

For this lab, part of your grade is determined by the quality of your unit tests. If you need a refresher on unit testing, please refer to the instructions from the previous labs.

1.6 A Sample Run

The `data/small.txt` file contains the following data. Each line represents a single toll event in the format `time,tag,exit` where the time is specified in minutes.

```
60,DR_J,45
61,I_AM_STROOT,45
62,BEN_KENOBI,45
100,I_AM_STROOT,39
150,DR_J,39
```

Upon execution with the above toll event data file, the main program should produce output matching the example below:

```
2 completed trips

On-Road Report
==============
[ BEN_KENOBI] on #45, time    62

SPEEDER REPORT
==============
Vehicle I_AM_STROOT, starting at time 61
    from Rochester - Victor - I-490
    to Syracuse - Fulton - I-690 - NY Route 690
    94.6 MpH

BILLING INFORMATION
===================
[       DR_J] on #45, time    60; off #39, time   150: $ 2.46
[I_AM_STROOT] on #45, time    61; off #39, time   100: $ 2.46
Total: $ 4.92

‘b <string>’ to see bill for license tag
‘e <number>’ to see activity at exit
‘q’ to quit
>
```

At this point, the user is free to enter one or more of the available commands. For example:

```
> b I_AM_STROOT
[I_AM_STROOT] on #45, time    61; off #39, time   100: $ 2.46

Vehicle total due: $ 2.46

‘b <string>’ to see bill for license tag
‘e <number>’ to see activity at exit
‘q’ to quit
> b DR_J
[       DR_J] on #45, time    60; off #39, time   150: $ 2.46

Vehicle total due: $ 2.46

‘b <string>’ to see bill for license tag
‘e <number>’ to see activity at exit
‘q’ to quit
```

```
> e 45

EXIT 45 REPORT
==============
[       DR_J] on #45, time    60; off #39, time   150
[I_AM_STROOT] on #45, time    61; off #39, time   100
[ BEN_KENOBI] on #45, time    62

‘b <string>’ to see bill for license tag
‘e <number>’ to see activity at exit
‘q’ to quit
> e 38

EXIT 38 REPORT
==============

‘b <string>’ to see bill for license tag
‘e <number>’ to see activity at exit
‘q’ to quit
> q
```

2 Submission

You will need to submit all of your code, including your tests, to the MyCourses assignment before the due date. You must submit your `hw6` and `hw6.test` folders as a ZIP archive named “`hw6.zip`” (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework). Moreover, the zip file must contain only the java files of your solution. Do not submit the .txt files provided in this homework nor compiled files.

3 Grading

The grade breakdown for this homework is as follows:

- Design 45%
  - General program design 20%
  - Choice of time-efficient data structures 25%
- Functionality 45%
- Testing 10%