Algorithms and Data Structures

Generics, Functional Interfaces and Recursion for Traffic Control

Assignment-2

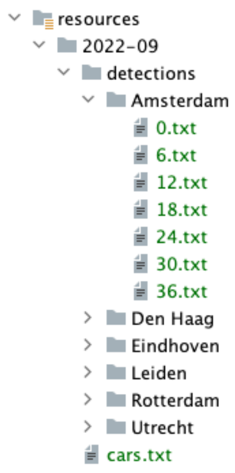
Version: February 10th, 2023

Introduction

At https://milieuzones.nl/ you find information about new legislation which restricts entrance of diesel fueled cars into in designated zones of city centers to protect quality of air. In this assignment you will work on an application that processes the logs of the detection gates at the entry points of those zones and aggregates the offences into fines for the car owners.

The key questions are:

1. Which are the top five cars with the largest total number of offences?
2. Which are the top five cities with the largest total number of offences?
3. What is the expected revenue from the fines which will be issued to the offenders?



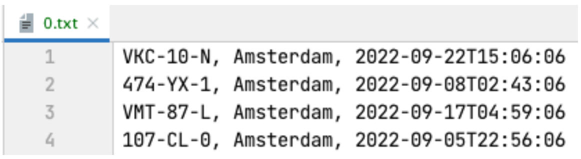
```
resources
  2022-09
    detections
      Amsterdam
        0.txt
        6.txt
        12.txt
        18.txt
        24.txt
        30.txt
        36.txt
      Den Haag
      Eindhoven
      Leiden
      Rotterdam
      Utrecht
    cars.txt
```

At the end of each month, every detection gate at an entry point of its zone uploads its log of observed cars into a central API on a vault of plain text files. The log files are organized within a hierarchical folder structure as shown in the picture at the left. Your solution shall run with direct use of this folder structure and shall be capable of dealing with thousands of these files organized in even more hierarchical levels of the files vault.

Format of the input data.

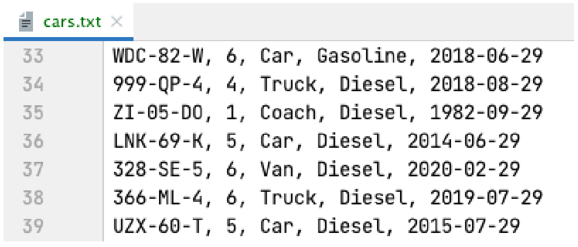
Each text file with detection information provides one line for each event that a car passes the detection gate. This line contains the license plate number of the car, the name of the city and a date-time stamp of the detection, each separated by a comma, as shown in the picture below.

There are no header data in the detection file and no other information. The lines can be in any order.



```
0.txt
VKC-10-N, Amsterdam, 2022-09-22T15:06:06
474-YX-1, Amsterdam, 2022-09-08T02:43:06
VMT-87-L, Amsterdam, 2022-09-17T04:59:06
107-CL-0, Amsterdam, 2022-09-05T22:56:06
```

Besides the vault with the detection data, also a 'cars.txt' file is provided that specifies relevant comma-separated information about all cars registered by the 'Rijks Dienst voor Wegverkeer' (RDW).



```
cars.txt
WDC-82-W, 6, Car, Gasoline, 2018-06-29
999-QP-4, 4, Truck, Diesel, 2018-08-29
ZI-05-DO, 1, Coach, Diesel, 1982-09-29
LNK-69-K, 5, Car, Diesel, 2014-06-29
328-SE-5, 6, Van, Diesel, 2020-02-29
366-ML-4, 6, Truck, Diesel, 2019-07-29
UZX-60-T, 5, Car, Diesel, 2015-07-29
```

The first value is the license plate number of the car.
The second value specifies its emission category. That is a number between 0 and 9. Higher numbers indicate cleaner cars.
The third field indicates the type of car.
The fourth indicates the type of fuel that the car uses.
The fifth is the date of registration of the car (also indicating its age.)

All cars have a unique license plate number. Their may be license plate numbers in the detection files that do not occur in the cars.txt file. These can be foreign cars or unregistered cars.

Access regimes.

Municipalities may implement different access regimes for different zones in their city. In this assignment we focus on the '<u>purple</u>' access regime. This regime prohibits **<u>diesel trucks</u>** and **<u>diesel coaches</u>** with an emission category of **<u>below 6</u>** to enter the purple zone. Other cars (of other type or with other fuel or with emission category of 6 or above) can enter freely. (Also unregistered and foreign cars can enter freely.)

In our data vault we have only included detection logs of purple zones.

Offending drivers will be fined. In this assignment we calculate the revenues when truck drivers will be fined €25 per offence, coach drivers €35 per offence.

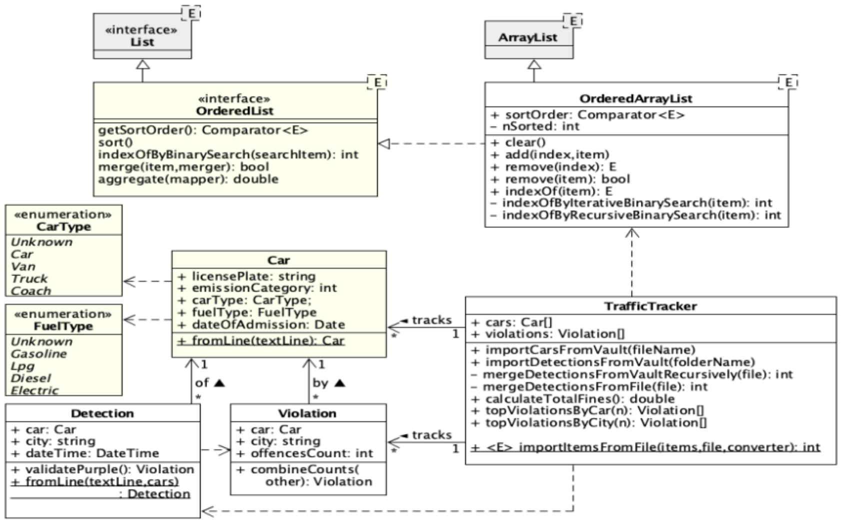
Solution approach.

Because we expect to use the detection data for other purposes in the future (other access regimes, other forms of taxation, speed control, etc.), we want you to invest in a maintainable solution with reusable software that can also be used to meet future requirements. The basic design has been documented in below class diagram. (Only the white classes need some coding from you.)

Car information will be captured in a Car class. For this assignment only the license plate, emission category, car type and fuel type are relevant.

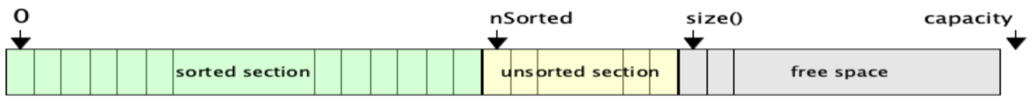
Detection information will be loaded and processed via the Detection class. The Detection class knows how to validate the entry against the purple access rules.

Offences will be registered by the Violation class. The Violation class can count multiple offences by the same car in the same city within a single Violation instance. When Detection.validatePurple() finds an offence, it will return a new Violation instance with offencesCount = 1.


«interface»
List
E
ArrayList
E
«interface»
OrderedList
E
getSortOrder(): Comparator<E>
sort()
indexOfByBinarySearch(searchItem): int
merge(item,merger): bool
aggregate(mapper): double
OrderedArrayList
E
+ sortOrder: Comparator<E>
– nSorted: int
+ clear()
+ add(index,item)
+ remove(index): E
+ remove(item): bool
+ indexOf(item): E
– indexOfByIterativeBinarySearch(item): int
– indexOfByRecursiveBinarySearch(item): int
«enumeration»
CarType
Unknown
Car
Van
Truck
Coach
«enumeration»
FuelType
Unknown
Gasoline
Lpg
Diesel
Electric
Car
+ licensePlate: string
+ emissionCategory: int
+ carType: CarType;
+ fuelType: FuelType
+ dateOfAdmission: Date
+ fromLine(textLine): Car
1
of ▲
*
1
by ▲
*
◄ tracks
1
*
TrafficTracker
+ cars: Car[]
+ violations: Violation[]
+ importCarsFromVault(fileName)
+ importDetectionsFromVault(folderName)
– mergeDetectionsFromVaultRecursively(file): int
– mergeDetectionsFromFile(file): int
+ calculateTotalFines(): double
+ topViolationsByCar(n): Violation[]
+ topViolationsByCity(n): Violation[]
+ <E> importItemsFromFile(items,file,converter): int
Detection
+ car: Car
+ city: string
+ dateTime: DateTime
+ validatePurple(): Violation
+ fromLine(textLine,cars)
: Detection
Violation
+ car: Car
+ city: string
+ offencesCount: int
+ combineCounts(
other): Violation
◄ tracks
1
*

To support these computations, we believe it is a good design to implement a generic extension 'OrderedArrayList' of the ArrayList class. This 'OrderedArrayList' maintains a sort sorder on all its items, such that its indexOf() can efficiently find an item by a binary search algorithm. It remembers the specific sort order being used in the **sortOrder** Comparator that was passed to the latest sort. That way you can also use that Comparator in your binary search implementations.

You may think, what benefit will I have from the binary search if I must re-sort the orderedArrayList each time when a new item is added. Hence, we introduce a representation invariant that allows for adding, inserting, and removing items without resorting:



In any state of the arrayList, only a first section has been sorted, and a (small, optional) part at the end is not sorted at all. The private instance variable **nSorted** keeps track of how many items are sorted. In the worst case, when nothing has been sorted yet, nSorted==0. In the best case, when everything has been sorted, nSorted==size().

The indexOf method now can use binary search on all items 0 <= index < nSorted (the sorted section), and if the item is not found it shall try a linear search across all remaining items nSorted <= index < size() (the unsorted section). The add method can add a new item at the end without additional sorting work. (Just extend the unsorted section with another item.)

If the unsorted part of the array has grown too large, the indexOf will be less efficient. It is up to the application that uses the OrderedArrayList to invoke a re-sort as appropriate. OrderedArrayList methods themselves may never re-sort automatically because then the client application could lose track of index positions of items which it may remember in local variables. Only the sort method itself will set nSorted = size() and clear the unsorted section.

Besides binary searching through its items, our OrderedArrayList can also merge new items into the list. If the merge operation finds an equivalent existing item in the list (when the sortOrder comparator yields zero) then the new item will be merged with the existing item using the provided 'merger' functional parameter. If no equivalent of the new item could be found, then the new item will be added to the list.

This functionality will prove very useful to aggregate the offencesCounts of multiple Violation instances according to different grouping criteria, e.g. group them by car or group them by city. A universal merger function for that has already been provided by Violation.combineOffencesCounts.

Another useful additional feature of our OrderedList is an aggregate method. This method should calculate some double value from each item in the list and add them all. We use this method to calculate the total revenue of the proposed scheme of fines for offences (€25 per truck, €35 per coach),

Assignment.

Unpack the starter project and proceed with implementing the missing parts:

1. Complete the implementations of the Detection and Violation classes:
   a) you need to complete the toString methods to obtain a proper text representation in output.
   b) you need to complete the static Detection.fromLine method which you will need for converting text lines of the source files into object instances.
   c) you need to complete Detection.validatePurple() to generate Violations for offending detections.
   d) you need to complete a comparator function to be able to order and merge Violations (and maybe later add other comparator functions for different groupings).
2. Complete the implementation of the generic OrderedArrayList class, which implements the OrderedList interface.
   a) Implement two versions of the binary search: an iterative version and a recursive version
   b) Implement the merge method and the aggregate method
3. Implement the missing parts of the TrafficTracker

4. You may add more private and public methods to any of the classes and interfaces, but not change any signature of the specified public methods.
5. Add unit tests for Detection.validatePurple(), TrafficTracker.calculateTotalFines(), TrafficTracker.topViolationsByCar() and TrafficTracker.topViolationsByCity() within two separate test classes DetectionTest2 and TrafficTracker2.
6. Also add additional unit tests (in separate test classes) for relevant code or situations that you did not manage to implement first time right and were not covered by the provided unit tests.
7. Run the main program and compare the results with below output.
8. Prepare your document with explanations of seven code snippets.
   Explain how the OrderedArrayList representation invariant is sustained by your overridden implementations of add(index, item) and remove(index).
   Involve the use of a loop invariant at least once when explaining correctness of a code snippet.

Sample partial output of TrafficControlMain

```
Welcome to the HvA Traffic Control processor

Imported 250 cars from 250 lines in /2022-09/cars.txt.
Imported cars:
[015-II-6/4/Coach/Diesel, 063-PF-3/5/Car/Gasoline, 084-AK-7/5/Van/Gasoline, 085-UD-5/5/Truck/Diesel, 088-XZ-
0/6/Car/Gasoline, 1-DLD-36/4/Car/Diesel, 105-OS-3/5/Car/Diesel, 107-CL-0/5/Van/Diesel, 109-JB-1/9/Car/Electric, 115-VX-
0/6/Car/Gasoline]...

Imported 1483 detections from …/2022-09/detections/Rotterdam/13.txt.
Imported 1884 detections from …/2022-09/detections/Rotterdam/7.txt.
Imported 2375 detections from …/2022-09/detections/Rotterdam/1.txt.
Imported 516 detections from …/2022-09/detections/Rotterdam/37.txt.
Imported 685 detections from …/2022-09/detections/Rotterdam/31.txt.
Imported 896 detections from …/2022-09/detections/Rotterdam/25.txt.
Imported 1159 detections from …/2022-09/detections/Rotterdam/19.txt.
Imported 1425 detections from …/2022-09/detections/Den Haag/14.txt.
Imported 492 detections from …/2022-09/detections/Den Haag/38.txt.
Imported 1812 detections from …/2022-09/detections/Den Haag/8.txt.
…
Found 7568 offences among detections imported from files in /2022-09/detections.

Aggregated offending detections:
[015-II-6/Amsterdam/43, 015-II-6/Den Haag/36, 015-II-6/Eindhoven/28, 015-II-6/Leiden/26, 015-II-6/Rotterdam/37, 015-II-
6/Utrecht/28, 085-UD-5/Amsterdam/42, 085-UD-5/Den Haag/35, 085-UD-5/Eindhoven/32, 085-UD-5/Leiden/33]...

Total fines à €25 per offence for trucks and €35 per offence for coaches would amount to: €207540
Top 5 cars with largest total number of offences are:
[69-WV-VS/null/225, E-737-HR/null/219, RMT-76-F/null/218, NUP-31-U/null/215, CKX-65-A/null/212]
Top 5 cities with largest total number of offences are:
[null/Amsterdam/1440, null/Rotterdam/1314, null/Utrecht/1295, null/Den Haag/1287, null/Eindhoven/1164]

Process finished with exit code 0
```

Grading.

At DLO you find the rubrics for the grading of this assignment. There are three grading categories: Solution (50%), Report (30%) and Code Quality (20%). Your Solution grade must be sufficient, before the grading of Report and Code Quality is taken into account. Similarly, your Report grade must be sufficient before the code quality grade is granted.