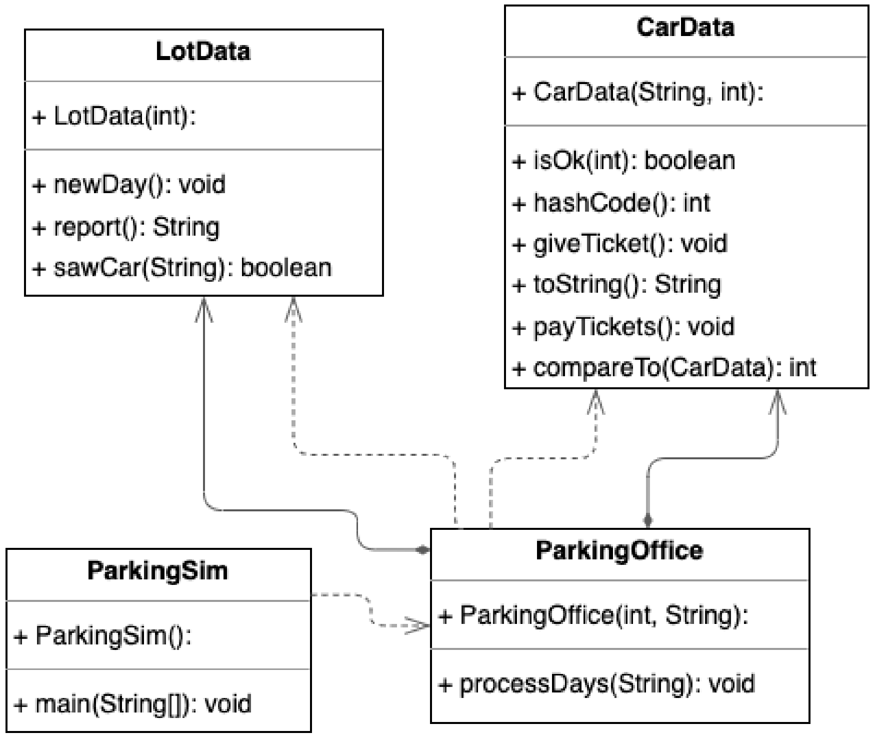
Computer Science AP/X — CSCI-140/242
Parking Office — Lab 5

1 Requirements

In this lab, you will have to build a system that will:

- Keep track of which parking lot each car is allowed to park in. (At this university, the parking lots are given different numbers, starting with lot 0, and each car is allowed to park in only one lot.) Cars are identified by their license plate.
- When a car is seen in a parking lot by the enforcement vehicle, check whether the car is allowed to park there, and if not, issue a ticket. A car can only get one ticket per lot per day, even if seen there multiple times, but if the same car parks in different lots illegally in one day, it will receive a ticket for each lot.
- At the end of the day, print out how many different cars were seen in each lot over the course of the day.
- At the end of each day, print out the ten cars with the most tickets.

2 Required functionality



2.1 Data classes

For this lab, you will need to complete the instantiation of `CarData` (which contains information about one car)and `LotData` (which contains information about one parking lot). The public methods have been shown, and you should use whatever data fields inside these that you find appropriate to create the required behaviors. In particular:

- The `LotData` constructor takes an integer representing the number of the parking lot.
- `LotData.report()` should return a String of the form `Lot 0 was used by 1 car(s) today.` (with the appropriate lot number and number of unique cars seen in the lot).
- `LotData.sawCar(String)` is called when a car is seen in that parking lot. It takes a String representing a license plate, records that information internally, and returns `true` if that car has **not** yet been seen in the lot that day.
- `LotData.newDay()` is called at the beginning of each day so that the class can update its data structure(s).
- The `CarData` constructor takes a String representing its license plate and an integer representing the lot it is allowed to park in.
- `CarData.giveTicket()` should increase by 1 the number of tickets assigned to that car.
- `CarData.payTickets()` should set to 0 the number of tickets assigned to that car.
- `CarData.isOk(int)` takes in the number of a parking lot and returns `true` if the car is allowed to park in that lot.
- `CarData.compareTo(CarData)`, `CarData.equals(Object)` and `CarData.hashCode()` should be created in order for your collections in `ParkingOffice` behave as required.

- `CarData.toString()` should return a String of the form `AAA1111 (lot 0) : 0 ticket(s)` (with the proper license plate, lot number and number of tickets).

Once you have completed these classes, the JUnit tests should complete successfully. Note that there are no other unit tests, instead you will need to use the event files provided (and probably some of your own!).

2.2 Events file

Within the simulation of the parking office (and enforcement vehicles), a number of different events can occur. These will be represented in a single text file that is read by your program. In particular, the event file may contain lines of the following types, for which you should implement the described behaviors:

- `BeginDay` - this will be a complete line of the file. Your program should reset any data structures as needed for the new day.
- `EndDay` - this will be a complete line of the file. Your program should print out a report similar to the following:
  ```
  End of day. Worst offenders are:
  ABC123 (lot 1) : 2 ticket(s)
  DEF345 (lot 2) : 1 ticket(s)
  Lot usage was:
  Lot 0 was used by 1 car(s) today.
  Lot 1 was used by 0 car(s) today.
  Lot 2 was used by 1 car(s) today.
  ```
  Note that if multiple cars have the same number of tickets, they should be printed out in lexicographical order (that is, the way that `String`s are sorted). You should print out at most 10 worst offenders, or fewer if there are fewer than 10 cars with outstanding tickets in the system (you should never print a car with zero tickets).
- Enforcement vehicle entering a lot - this will simply be represented by an integer on a line by itself, where the integer is the lot number being entered.
- Payment office - this is represented by `P` on a line by itself, to start the list of cars paying off their tickets.
- Car seen - any line of the file that does not match any of the above represents by a license plate . This is either a car being seen in a parking lot, or the owner paying the car's ticket - which one of these is determined by the most recent non-car line of the event file. That is, if we saw a lot number, then cars, those cars were seen in that lot, but if we saw a `P` and then cars, those cars were paying their tickets.

2.3 Sample Event File

This file is included as `test1-events` in the zip file. It should be used with `test1-cars` and three parking lots.

```
BeginDay
1
ABC123
```

```
0
DEF345
EndDay
BeginDay
2
ABC123
GHI567
0
ABC123
EndDay
BeginDay
P
ABC123
2
DEF345
EndDay
```

2.4 Sample Output

This is what the output should look like for the given file.

```
------------
End of day. Worst offenders are:
DEF345 (lot 2) : 1 ticket(s)
Lot usage was:
Lot 0 was used by 1 car(s) today.
Lot 1 was used by 1 car(s) today.
Lot 2 was used by 0 car(s) today.
------------
End of day. Worst offenders are:
ABC123 (lot 1) : 2 ticket(s)
DEF345 (lot 2) : 1 ticket(s)
Lot usage was:
Lot 0 was used by 1 car(s) today.
Lot 1 was used by 0 car(s) today.
Lot 2 was used by 2 car(s) today.
------------
End of day. Worst offenders are:
DEF345 (lot 2) : 1 ticket(s)
Lot usage was:
Lot 0 was used by 0 car(s) today.
Lot 1 was used by 0 car(s) today.
Lot 2 was used by 1 car(s) today.
```

3 Hints and Tips

- Note that the sample file does not cover all possible situations — for example, the enforcement vehicle may enter the same lot at multiple times throughout the day. We have provided an additional test file that includes this possibility, as well as some test files with much larger data sets. However, even these may not cover every possiblity.
- You can assume that all input files are properly formatted according to the above, and that no cars will appear in an event that were not included in the file of cars.
- As noted in the problem solving, for reasonable cases, it is much more efficient to keep a collection of `CarData` objects (only those with tickets, sorted by number of tickets) up-to-date as each ticket is issued, so you should proceed in this fashion in your `ParkingOffice`
- Recall that if you modify an item within a sorted or hashed collection, the collection will not notice! (This is why Python requires dictionary keys to be immutable types, so that you can't do this by accident.) So if you have such a collection and want to modify an element of the collection, you will need to remove the data item in question, modify it, and re-add it. (Note that this is not an issue if your key or set element is of an immutable type.) While this is annoying, note that it is still a constant number of operations so should be the same overall time complexity as simply adding to the collection (whatever collection it is).

4 Grading

- 15%: Attendance at problem-solving and results of problem-solving teamwork
- 25%: Design: appropriate collections used for the requirements, appropriate modularity/encapsulation
- 55%: Functionality
  - 15%: CarData
  - 10%: LotData
  - 25%: ParkingOffice (file parsing and event handling)
  - 5%: ParkingSim
- 5%: Good coding style

5 Submission Instructions

Zip up your entire `src` folder (not the entire project folder!) and name it `lab5.zip`. Upload and submit it to the MyCourses dropbox before the due date. Note that due to the break, the lab is due end of day Friday instead of the normal Wednesday.