MCIT 594 – Group Project Report

Sean Bivins, Federica Pelzel

Additional Features

1. Fines to Average Market Value Ratio

Our additional feature for this project was a method that calculates the ratio of total fines to average residential market value (RMV) for a given zip code. We wanted to see if there was a constant relationship between the economic status of a zip code (measured by RMV) and the total amount of fines collected in that zip code. The hypothesis that we had was that some "poorer" zip codes would end up having more fines collected than the "richer" ones (e.g. San Francisco). We found this to be the case in some of the poorer zip codes of Philadelphia—such as 19104, 19132, and 19141—where their RMVs were on the lower side, but the number of fines was the highest.

2. Get Parking Violations for a specific Car ID

The user will input a car ID, and the program will return a description of all parking violations and total fines due for that car. In order to make this feature more efficient cars were not only added to the ParkingViolation object and subsequently to the ZipCode, but they were also added to a HashMap of Car elements with the CarID as keys. This allows for a more efficient iteration when searching for a specific Car, with O(1) complexity.

Output example:

```
7
Enter a 7-digit car ID to get Parking Violations (example: 1199878)
1687997
1. Timestamp: Tue Jan 22 15:15:00 EST 2013 ViolationID: 2906043 Description: METER EXPIRED CC Fine due: $36
2. Timestamp: Wed Jan 02 14:19:00 EST 2013 ViolationID: 2906046 Description: METER EXPIRED CC Fine due: $36
3. Timestamp: Thu Jan 24 09:10:00 EST 2013 ViolationID: 2906038 Description: STOP PROHIBITED CC Fine due: $76
4. Timestamp: Fri Jan 11 13:22:00 EST 2013 ViolationID: 2906040 Description: METER EXPIRED CC Fine due: $36
5. Timestamp: Sat Jan 19 13:17:00 EST 2013 ViolationID: 2906048 Description: OVER TIME LIMIT CC Fine due: $36
6. Timestamp: Fri Jan 11 13:15:00 EST 2013 ViolationID: 2906041 Description: METER EXPIRED CC Fine due: $36
7. Timestamp: Fri Jan 25 15:34:00 EST 2013 ViolationID: 2906045 Description: METER EXPIRED CC Fine due: $36
8. Timestamp: Fri Jan 18 11:14:00 EST 2013 ViolationID: 2906042 Description: METER EXPIRED CC Fine due: $36
9. Timestamp: Mon Jan 14 11:16:00 EST 2013 ViolationID: 2906036 Description: PARKING PROHBITED CC Fine due: $51
10. Timestamp: Sat Jan 05 10:53:00 EST 2013 ViolationID: 2906047 Description: METER EXPIRED CC Fine due: $36
11. Timestamp: Wed Jan 30 21:56:00 EST 2013 ViolationID: 2906039 Description: STOPPING PROHIBITED Fine due: $51
12. Timestamp: Tue Jan 08 10:57:00 EST 2013 ViolationID: 2906044 Description: METER EXPIRED CC Fine due: $36
13. Timestamp: Thu Jan 03 20:01:00 EST 2013 ViolationID: 2906049 Description: OVER TIME LIMIT Fine due: $26
14. Timestamp: Wed Jan 16 13:27:00 EST 2013 ViolationID: 2906035 Description: METER EXPIRED CC Fine due: $36
15. Timestamp: Mon Jan 14 09:51:00 EST 2013 ViolationID: 2906037 Description: SIDEWALK Fine due: $51

Total due for car ID 1687997: $615
```

Use of Data Structures

Three data structures we used in our program were:

TreeMap<Integer, ZipCode>

When analyzing the instructions, we realized that most methods in the data analysis requirements start by calling on a ZipCode. To that effect, we decided to create a TreeMap that holds the zip code number as the key, and a ZipCode object as the value. TreeMap get/put/containsKey() operations are between O(logN).

We also chose a TreeMap even though it's a little bit slower than other maps (like HashMap which has O(1) get/put/containsKey()), because of its sorted nature, and the fact that it would provide us with better memory performance.

HashMap<Integer, Car>

For additional feature 2, we decided to use a HashMap that allowed us to much more quickly and efficiently retrieve a car by its ID and return its associated Parking Violations. With O(1) complexity for get/put, HashMap was an ideal choice for this feature, given it didn't require for cars to be sorted in any order. Within each car object we used another data structure, HashSet, to store the ParkingViolations related to that car. We chose HashSet for this because it would have faster performance than an ArrayList.

HashSet

ZipCode objects hold HashSets of ParkingViolations and Properties, that allow for efficient add() and contains() methods.