CIT 594 Project Report

Wes Poulsen and Garret Bassett

Additional feature

The additional feature we implemented calculates the fines per capita and market value per capita for each ZIP code, then ranks the ZIP codes across those two metrics and creates a condensed plot to illustrate the relationship between the two.

This feature uses the `parking.csv` (or `parking.json`) file to calculate total fines per ZIP code, the `properties.csv` file to calculate total market value per ZIP code, and the `population.txt` file to convert the previous two metrics into per-capita.

We tested this function by performing the separate calculations on the dataset in Excel and plotting the points. Since this graph is condensed (the axes are 25% as wide as the number of possible values), multiple values may be represented by a single "*", which it does correctly.

Data structures

Our program uses the following data structures:

1) **HashMap.** This structure is used multiple times throughout the program, wherever there is a need to reference a group of attribute values connected to a single key. It is found in the following classes / functions:
    a. **Reader classes.** HashMap fits here because it allows us to read rows of data and attach them to a single object (Violation, Zipcode, or Property). We considered TreeMap for ordering but decided to implement TreeMaps later where needed depending on what ordering was required.

    b. **DisplayFinesVsMarketValues (in Processor Class).** We implemented a HashMap here in order to attach both a total fines and total market value metric to the same ZIP code. We considered instead including those as instance variables in the ZipCode class, but it seemed like bad design to have an instance variable in a Data class that was only updated from the Processor layer.

    c. **AverageByZip.** The interface where we implemented the Strategy Pattern takes in a HashMap of unique ID keys and property values, as well as a ZIP code; it loops over the properties and compares them to the ZIP code argument. This could have been an ArrayList, but the unique ID keys were helpful in debugging.

2) **TreeMap.** This was implemented in displayFinesPerCapita, since that was the easiest way to sort keys in ascending order. Originally it was a HashMap but that needed to be sorted in a second step, so the TreeMap was easier.

3) **ArrayList.** We used an ArrayList to keep track of column names in the properties.csv file. We used an ArrayList rather than an Array in order to keep sizing dynamic in case we wanted to add in additional fields later.

4) **Array.** Arrays were used when a finite, known set of values needed to be stored in a repeated manner. In addition to being used by default whenever String.split() is used, an Array is also used in **DisplayFinesVsMarketValues**, with each ZIP code key holding an Array of four values (total fines, total market value, fines per capita, market value per capita). We considered an ArrayList, but an Array has better performance and is less verbose when referencing an element by index.