Ankita Patel
Adrien Alexandre

1. Project report, CIT594 – Final Project, Team 131

The purpose of this document is to give a brief overview of: 1) the additional custom feature that team 131 built for user input 7, 2) describe the use of three data structures that we have used during the course of this assignment, and 3) share our learnings and how we may apply them in the future.

2. Custom Feature Description

For our custom feature: when the user selects option 7, the user gets prompted to enter a comma delimited list of zip codes. The program then outputs that same list to the user, but with two extra fields: 1) Maximum Hospitalization Rate per capita achieved for that zip code, 2) Average Market Value of properties for that zip code). The list of Zip Codes is provided in ascending order. For instance if the user inputs 19100, 19103, 19128, the resulting list would look like (Made up numbers):

BEGIN OUTPUT
19100 0.2322 876453
19103 0.1432 1001253
19128 0.0566 745654
END OUTPUT

If a ZIP code doesn't exist or does not contain any covid data or property data, then it does not get displayed.
If no ZIP codes can be displayed, then the program prints "0".

In order to unit test this feature, we performed various tests by analyzing the expected result for a sample of ZIP codes (we computed 10 individual zip codes and their expected results), and then verifying that the output was the same as the one we expected. Unit testing was performed individually by both Adrien and Ankita, to ensure higher coverage of testing.

3. Description of Data Structures

Our code was largely structured around the use of a custom "ZIPCode" object (See appendix). Each ZIPCode object contains a number of attributes:

1) an int storing the ZIP Code (e.g. 19100).
2) an int storing the ZIP Code's population, as extracted from the population file.
3) a HashMap<String, HashMap<String, Integer>> to store the COVID Data for that zip code.
4) a LinkedList<String[]> to store a pair of values (market value, total liveable area) from the properties file.

When we "read" our files, the readers populate (or updates) the zip code information for each zip code it finds, and the zip codes are then stored in a hashmap contained within a singleton instance of ZIPCodeDataHandler (to ensure that we only work with only one zipcode hashmap).
Below, we describe the use of 3) and 4) in our attribute list above. We also describe the use of a TreeMap<Integer, Double> returned by the processor tiers, used to display vaccination rate per capita, ordered by ascending zip code.

3.1. COVID Data, stored inside a nested HashMap

In order to make the process of storing and retrieving covid data easier for one specific ZIP, we decided to use a hashmap, with the date string as key, and a nested hashmap (key: column name, value: value) as the value. With this combination, we are emulating the concept of a json object using custom data structures. An example of what this map contains would be:
{'202-03-25': {'POS': 23, 'NEG': 58}}
The reason we used this system was: O(1) complexity for adding onto the map, and O(1) complexity for retrieving, given a specific key. Since the user inputs both a date and a vaccination type, it was very easy for us to retrieve what the user is asking for in O(1) time.

3.2. Properties data, stored as pairs of strings inside a Linked List

The choice of a Linked List here was largely driven by 1) the fact that we did not need to retrieve properties by index, we only care about their value and livable area, 2) the extremely large file (500k+ rows), which means we need fast inserts in order to store that data. An ArrayList has a worst case scenario of O(n) for adding an element, so a linked list made sense here. Each node of our Linked List contains a pair of integer values (String array of size 2) corresponding to Market Value and livable area. Since we have to iterate through the whole list anyway in order to calculate the total livable area and market value, but do not have to retrieve elements by index, a linked list made the mode sense due to the very fast append operations.

3.3. Vaccination Rates per Capita Per ZIP returned as a TreeMap from processing

Since we need to display the output as an ascending list of vaccination rates, by order of Zip Code, a TreeMap containing Zip Code as Key and a double representing vaccination rate for that zip code made perfect sense. This allowed us to print the resulting map to the user in a much faster time than if we had to order a hashmap or other type of data structure.

4. Learnings

4.1. Data Structures

In the spirit of being self-critical: after seeing week 14's lecture, we realized that for the properties data that we stored as a Linked List, an ArrayList may have made more sense to allow for parallelization of the addition and count operations, which are the only processing done on Market Value and Livable Area. Operations are more difficult to parallelize inside a linked list, since retrieving the middle point of the linked list would be O(n) in time. We haven't considered this much further, since our code runs fast enough, but it's something we could optimize if we wanted to refactor later.

4.2. Project Collaboration

In order to collaborate for this project, we scheduled regular Zoom Meetings every two days for a week, while we were making progress on areas of the code. We split the work mostly by Tiers (which is one of the benefits of functional independence and the N-Tiers architecture, no dependencies on other code so long as the interface is agreed upon!). Ankita worked on the Processing and UI tiers while Adrien handled the Data Management and Utils tiers. We shared code back and forth by email, which is sub-optimal. It worked thanks to how well the N-tiers allows us to organize code, but this method would not work in a professional setting. Using GitHub will become essential for handling larger enterprise-scale projects in the future, and is something we will need to learn and become better at.

5. Appendix: UML Diagram (Simplified)

The UML diagram below presents the overall structure of our program, but does not contain all of the methods we used (it would be much messier otherwise).