**594 Final Project Report**

**Team 041: Shuo Li, Wenjing Wu**

1. Additional Feature.

When user inputs "6", they will be firstly asked to type 'partial' or 'full" for partial/full vaccinations data, then they will be asked to enter 'highest' or 'lowest' to search for town with the highest/lowest vaccination rate. The program will check for user input until they enter as required (case is insensitive).

The **void printVaccAreaInfo(boolean partialFlag, boolean highestVaccFlag)** function takes the two user inputs and search for the zipcode that has the highest/lowest partial/full vaccination rate. Then it prints out the town's population and property information (average property market value and average livable area). Note a town’s property market value and average livable area information can be retrieved directly from the maps **avgMarketValueMap** and **avgLivableAreaMap** that has might have saved information in #3 and #4. This is an application of memorization to avoid duplicated computations.

The correctness is guaranteed as we compared with the three input files to make sure the outputs are consistent with the raw records.

1. Use of Data Structures
2. ArrayList(array)

* Where: In several Reader classes under datamanagement package, we use ArrayList to store information read from the input files. For example, we use List<Population> populationList to store data read from population.txt file, Population is a new object type we created to store all information covered in one record, which include a zip code and the corresponding population
* Why: In Java API, ArrayList uses a dynamic array, it’s easy to resize. Since the Reader class reads different files with various number of data, ArrayList is convenient to store data not limited to a specific size.it’s very easy to access and iterate. Accessing any element by index takes constant time O(1). We can easily check the data stored in a particular position. The order of the elements in ArrayList is always kept same as the order they are added; ArrayList can store all types of objects, this makes it convenient for us to create new object class and store in ArrayList;
* Which alternative is considered: LinkedList and HashSet.
* Compared to ArrayList, LinkedList is efficient to manipulate its elements. It’s efficient to insert or delete, however, we have no need to manipulate data when we read data, the function of Reader class is only to get and store all data.
* One big reason that not using HashSet is it does not allow duplicates, however we may read duplicate data from input files, not allowing duplicates will result in inaccurate reading of data.

1. TreeMap(Red-Black Tree)

* Where: In covid processor, we use TreeMap to store pairs of values <zipCode: number of partial or full vaccination per capita>, which is feature 2 in the instruction pdf.
* Why: TreeMap keeps its entries sorted according to the natural ordering of its keys. When key is zipCode, pairs will be added and kept in the ascending order of zipCode.
* Which alternative is considered: HashMap and sort.
* Different from TreeMap, HashMap is a hashtable based implementation. For basic operations get() and put(), HashMap takes O(1) which is faster than TreeMap taking O(logn). Store data in HashMap without order and then sort the key by ascending order will achieve the same result for feature 2, but we do not choose HashMap because TreeMap code looks cleaner and we don’t want to waste time sorting HashMap for everytime we run feature 2.

1. HashMap (Hash table)

* Where: In Population processor, we use HashMap to store pairs of values <zipCode: population> in function getPopulationMap() and getPopulationByZipcode(). These 2 functions are used in feature 2, 5 and 6.
* Why: HashMap is faster and easy to store pairs of values. In feature 2,5 and 6, they all have 1 step to get the total population by its zipcode. Since previously we store such data pairs in HashMap, we can easily access the population(value) of a specific zipCode (key) by calling get() function of HashMap. Although HashMap does not keep order, we do not care about order when we already know which key’s value we are looking for.
* Which alternative is considered: ArrayList.
* We can achieve the same result by using ArrayList. We can iterate through each object (Population) in ArrayList and check its field (Population.getZipCode( )), if such field is what we are looking for (Population.getZipCode( ) = target zipCode), we can return other fields we want from this project (Population.getPopulation( )). However, this kind of implementation is not as efficient as HashMap since it only takes O(1) for HashMap to get value of a key, it will at worst takes O(n) for ArrayList to get value of a key.