COMP9103 Pet Ownership Statistics System

Andrew Thia, 480568801

Overall Design

The software is made up of four classes:

- Resident
- ResidentDatabase
- ResidentProcessor
- POS

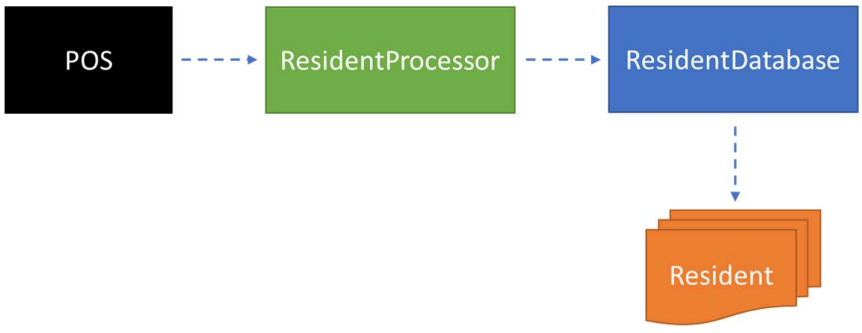
The *Resident* class stores information for a single resident. It contains methods to update, verify, retrieve and compare the details of the corresponding resident.

The *ResidentDatabase* keeps a database of all residents. The information of each resident is kept in a *Resident* class instance. The *ResidentDatabase* also has methods to query, sort and update the database.

The *ResidentProcessor* class parses the specified input files (records and instructions) and delivers records/instructions to the *ResidentDatabase* for processing. It also obtains results from the *ResidentDatabase* and saves them in specified output files.

Finally, the *POS* class implements a simple command-line interface so the user can invoke the Pet Ownership Statistics system through Command Prompt. Arguments provided by the user are passed on to the *ResidentProcessor* class for actual processing.

A simple diagram of how the classes interact is given below.



For more information on the available fields and methods for each class, see the generated Javadoc in the ***doc*** folder.

System Notes

Below are some notes about some specific behaviour of the system.

Resident field values

- Name values are case-insensitive, they get standardised with proper case when stored in the *Resident* class.
- Two-digit year specifications for birthday is supported. If the two-digit year is below 19, the year is inferred to be after 2000. If the two-digit year is greater than or equal to 19, the year is inferred to be between 1900 and 2000.
- An address specification must include suburb to be valid. A suburb is assumed to be given if one of the address fields (separated by comma or new line) is all made up of all characters and has a character length greater than 3 (to differentiate from state).
- The name of pet types is case-insensitive. Pet types are also standardised and stored with proper case by the *Resident* class.
- An owner may have more than one pet of the same pet type. This can be specified by listing the same pet type more than once, eg: *pet dog dog dog cat cat*.
- The software tries to differentiate between the owner having no pets and the owner having an unknown number of pets. If the pet keyword is specified but followed by blank values – then it is assumed the owner has no pets. But if the pet keyword is not specified at all, then the number of pets and whatever possible pets the owner may have is treated as unknown. In any case, the behaviour of both cases (no pets or an unknown number of pets) is the same when queried.

Database unique identifier

- The *ResidentDatabase* class uses a resident's name and phone number as a unique identifier. This means no two residents on the database can have the same name and phone number combination. If the input records file has multiple records with the same name and phone number, the database will load the information of the first record into the database, then use subsequent records of the same name and phone number to update the existing entry in the database.
- The program is not smart enough to determine slight variations in name for the same person. For example, if the same person is listed twice with the same phone number but the first listing includes the person's middle name and the second listing does not, the program will treat both records as two distinct persons.

Queries

- Instructions are processed in order of their listing on the instructions file. As such, queries are performed on the "as at" state of database that reflects the order of the query relative to the rest of the instructions. This means, any instructions that are listed after a particular query (eg. update or delete resident) will not be reflected that query's results.
- When querying records, if no records exist at the time of the query, the system will still print the query header in the output report file, eg:

  ```
  ------query name Tom White------
  --------------------------------
  ```

  This is a deliberate behaviour and aims to give the user an indication that the query was indeed performed but no records with a matching name were found; as opposed to printing nothing at all which does not allow differentiation between 1) the query was somehow not performed, or 2) the query did not find any matching records.

- The age profile query calculates age values using the "age last birthday" method relative to the current system date.

Testing

Testing of the system is conducted using two test classes, the sample test cases provided and some additional test cases. The two test classes focus on testing the *Resident* and *ResidentDatabase* classes. While test cases evaluate whether the system as a whole works as intended. The source code of these test classes can be found in the ***src*** directory (together with the source code of the other classes). Test cases are located in the ***test*** directory.