CS 245 - Assignment 2 Summer, 2021

Assignment 2 - What's My Age?

Imagine Americans named Madison, Sydney or Hailey. You might envision people with these names as teenage girls. Now imagine people named George, Abe or Alfred. These names are probably more associated with older men. As it turns out, the Social Security Administration (SSA) has been keeping track of the names of people born in the United States and they have released this data to the public for academic use.

The goal of this assignment is to generate the likely age associated with a provided name, gender and state. In so doing, your implementation will show mastery of classes, containers and polymorphism in Java.

Background

The US Social Security Administration has a record of all names of children born by state and year, available via https://www.ssa.gov/oact/babynames/state/namesbystate.zip. Unzipped, this becomes 50 files — one for each US state according to its state code — plus one file for the District of Columbia (DC) and one "read me" PDF file. For reasons of privacy, some names are not included in these data files, specifically when there are fewer than 5 children born in a combination of state, gender and year with the name in question.

Names from these data files tend to show a temporal effect. For example, the name "David" in California was most popular in 1961, according to the data extracted from these files, as shown in Figure 1.

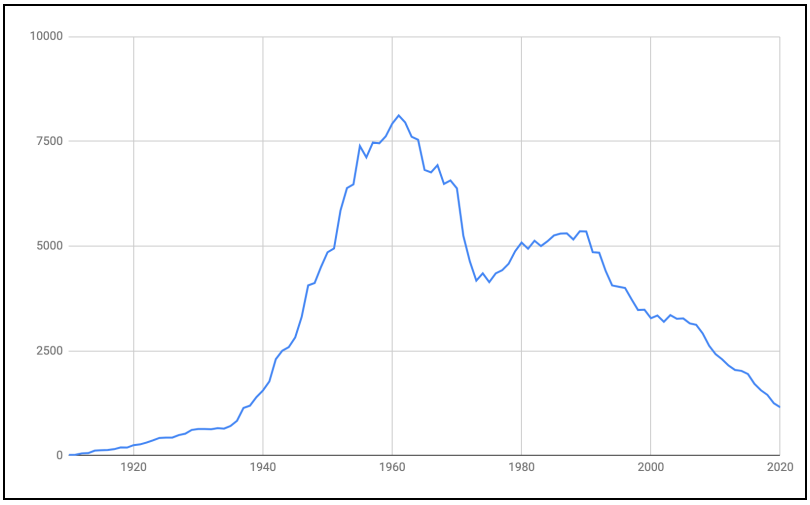

Figure 1: Number of male children named "David" born in California by Year Born

From the California “David” data, it is reasonable to assume anyone named “David” in California was likely born in 1961 and is therefore approximately 60 years old[1], given the current year is 2021.

Requirements

Your implementation will read the data files provided, store the data using either an ArrayList or a LinkedList instance and allow querying or reporting on this data. Details of each of these follow. Your implementation will be provided with the name of a single directory / folder containing one or more of the SSA files available via the link above.

Requirement 1: Your implementation will be provided a single property / configuration file as the first argument. This file will be in key=value format and will have at least the keys “ListType” and “Directory” as show in the example below:

```
ListType=ArrayList
Directory=/Users/dbrizan/namesbystate
```

These properties may appear in any order. The property ListType indicates the type of List which is required for your implementation. Two values — `ArrayList` and `LinkedList` — are valid. You may provide implementations of other container classes if you choose. Any accessible directory (folder) is valid for the “Directory” key. This directory contains one or more of the files from the SSA. The directory may only contain these files and no other data. If the configuration file is not provided or is missing or if any of the key-value pairs is improperly specified, your implementation must exit, gracefully, with a message indicating the reason for early termination.

Requirement 2: Your implementation must store the data from the SSA sufficient to perform the queries in the data below. Specifically, your implementation must use either an ArrayList or a LinkedList implementation, as specified in the configuration file, detailed above. Your implementation must use your own ArrayList or LinkedList classes. (Your implementation must NOT use these classes from the java.util package.)

Requirement 3: Once the data has been read and stored, your implementation must accept one or multiple queries from the user and, where possible, must respond with the likely age of the person being queried. An example of this interaction is as follows, with the implementation’s outputs in bold and the user’s inputs in non-bold font:

```
Name of the person (or EXIT to quit): Chloe
Gender (M/F): F
State of birth (two-letter state code): NY
Chloe, born in NY is most likely around 11 years old.
Name of the person (or EXIT to quit): EXIT
```

The calculation of a girl / woman named Chloe being about 11 years old is based on the current year being 2021 and the most popular year for the name Chloe in the state of NY being 2010. Your implementation must deal with atypical or unsuccessful paths gracefully. For example:

---

[1] To be clear, my first name is not “David,” I was not born in California (or anywhere else in the USA), and I was not born in 1961. But I may have given away someone else’s age. Perhaps. (Whoops!) Maybe not.

- If the user inputs any invalid data (eg. “AU” for the state code), your implementation must prompt the user for valid entries.
- If the query results in no age hypothesis, your implementation must indicate so and begin prompting for another name, gender and state combination.
- If the query results in multiple valid age hypotheses, your implementation may indicate an age range instead of a single age.

Recommendations

Recommendation 1: Use Properties (java.util.Properties) to read the property file — see Requirement 1 — and to get values from the file via your Properties instance.

Recommendation 2: The data files provided by the SSA are in CSV format, despite the name. (There is a description of the CSV file format in many places, including at Wikipedia. In this case, the file has no strings containing commas, so there should be no quote characters.) While not required, you may use a CSV reader for this. There are several CSV readers, including com.opencsv.CSVReader. (You’ll need to include the source — ZIP or JAR file? — in your build path.) An example at TutorialsPoint shows how to use CSVReader from OpenCSV. If you use a CSV reader, include details in a README file.

Submission

Submit the link to your GitHub repository or the source code for your implementation on Canvas. You may also add any comments in a README file (text, PDF or Word document) to help the grader understand or execute your implementation.

Grading

Your grade for this assignment will be determined as follows:

- 60% = Implementation: your class implementations must run successfully with a subset or superset of the example data available. It must produce the expected results, a sample of which appears in the Requirements section above. Any deviation from the expected results results in 0 credit for implementation.
- 20% = Decomposition: in the eyes of the grader, the degree to which your solution represents a reasonable object-oriented / procedural decomposition to this problem.
- 10% = Efficiency: In the eyes of the grader, the degree to which your implementation makes choices to minimise the running time or memory requirements.
- 10% = Style: In the eyes of the grade, the degree to which your code is readable to the point of being self-documenting. Specifically, names for variables and functions must be descriptive. Any code which is not straightforward or is in any way difficult to understand must be described with comments. Classes and functions should be commented using Javadoc.