Joanna Klukowska
joannakl@cs.nyu.edu

Project 5: Baby Names Revisited

Due date: April 28, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will revisit the problem from project 1. The goal of this project is to provide a more efficient implementation. The user experience should be very similar to the one from project 1.

Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- working with multi-file programs
- reading data from input files
- using command line arguments
- working with large data sets
- implementing an AVL tree
- writing complex code projects
- reusing previously written code

Start early! This project requires significant code to be written.

For the purpose of grading, your project should be in the package called `project5`. This means that each of your submitted source code files should start with a line:

```
package project5;
```

Dataset

In this project you will use the same dataset that you used in project 1, the NYS Baby Names database. Revisit that first project for details of how the input file is organized.

User Interface

Your program has to be a console based program (no graphical interface) - this means that the program should not open any windows or dialogs to prompt user for the input.

Program Usage

The program usage (use of command line arguments) is exactly the same as in project 1. You should be able to reuse large parts of your own project 1 code if it was implemented correctly.

Joanna Klukowska
joannakl@cs.nyu.edu

Input and Output

The program should run in a loop that allows the user to check popularity of different names in different counties. On each iteration, the user should be prompted to enter a name and a county (for which the program computes the results) or the letter 'Q' or 'q' to indicate the termination of the program.

The user should not be prompted for any other response.

If the user enters 'Q' or 'q' for the name, the program should terminate.

If the user enters 'ALL' for the county, the program should compute the results for the given name for the entire state of NY (all the counties).

If the name/county combination entered by the user cannot be found in the list of names stored in the dataset, the program should print a message

```
No such name/county in the dataset.
```

to the output stream and the program should continue into the next iteration.

Histogram format:

If the name entered by the user is found in the dataset for at least one year, a histogram showing the popularity of this name should be displayed.

For each year's data, the program needs to determine the percentage of babies in the specified county with the particular name

$$\frac{\text{number of babies with a given name}}{\text{total number of babies in a given county in that year}} \times 100$$

For each year there should be a line of output matching the following format:

```
YYYY (F.FFFF): HISTOGRAM_BAR
```

where

- `YYYY` is a four digit year,
- `F.FFFF` is a percentage calculated according to the above formula (it has to be printed with exactly one digit before the decimal point and four digits after the decimal point[1]),
- `HISTOGRAM_BAR` is a visual representation of the percentage; it should consist of a sequence of vertical bars '|', one bar for each 0.01 percent (rounded up to the nearest integer); the number of bars can be calculated by

  $$\left\lceil \frac{\text{number of babies with a given name}}{\text{total number of babies in a given county in that year}} \times 10000 \right\rceil$$

  where the symbol $\lceil \ldots \rceil$ means the ceiling function (in Java you can use `Math.ceil()` to compute it, see https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html#ceil-double-)

Here are the example lines calculated for 'Joanna' for three different years:

```
2007 (0.0583): ||||||
...
2010 (0.0515): ||||||
...
2015 (0.0320): ||||
```

(Note, that the program should not be printing ..., but rather the information for each year.) There should not be any blank lines between the lines for each year.

A few things to consider:

- For a name that does not occur in a particular year, the count should be set to zero, which will result in no vertical bars printed.
- For a name that is the most popular in a given year, the number of bars may exceed the width of the display window and (depending on the settings) may wrap to the next line. This is fine and you do not need to modify this behavior.

[1] You should be using `printf` for that. If you are not familiar with formatted output in Java, research it or ask questions.

Joanna Klukowska
joannakl@cs.nyu.edu

- Some names may occur in the data file more than once (since they are given to both female and male babies). Your histogram should combine the data for both occurrences.
- The program should be case in-sensitive. The name in the data file is always capitalized, for example 'Joanna'. Your program should produce exactly the same results regardless if the user types 'joanna', 'JOANNA', 'JoAnNa' or any other variation of cases.[2]
- Some names have several different spellings in common use, for example 'Joanna', 'Johana', 'Johanna'. For the purpose of this program these are considered to be completely different names.
- Any lines in the input file that do not contain the required five fields (year, name, county, gender and count) should be silently ignored and the program should move on to reading the next line from the file.

A few sample interactions are shown at the end of this document.

Data Storage and Organization

Your need to provide an implementation of several classes that store the data and compute the results when the program is executed.

In particular, your program must implement and use the following classes. You may implement additional classes as well, if you wish.

`Name` Class

The `Name` class is should be very similar to the one implemented in project 1. The major change is that the `Name` objects should keep track of the county as well.

The `Name` class stores information about a particular name for a particular county. It should store information about the name itself, the gender, the count (how many babies have been given that name) and the county in which the babies with a given name were born.

This class should provide a four parameter constructor:

```
public Name ( String name, String gender, int count, String county )
```

If the constructor is called with an empty string for `name`, invalid `gender` indicator (valid values are single characters 'f' for female, 'm' for male in either lower- or uppercase), a negative value for `count`, or an empty string for `county` name , then an instance of `IllegalArgumentException` should be thrown carrying an appropriate error message. None of the parameters to this constructor are allowed to be `null`. If the constructor is called with a `null` parameter it should throw an instance of `IllegalArgumentException`.

There should be no default constructor.[3]

This class should implement `Comparable<Name>` interface. The comparison should be done by the `name` as the primary key (using alphabetical order), by the `county` as the secondary key, by the `gender` as the tertiary key (i.e., when two objects that have the same value of `name` and `county` are compared, the comparison should be performed by `gender`).

This class should override the `equals` methods. The two `Name` objects should be considered equal if the `name`, `county`, and `gender` data fields are identical.

The class should override the `toString` method. The details are up to you, but you should make sure that it returns a `String` object that is a meaningful representation of the object on which it is called.

`YearNames` Class

The `YearNames` class should be used to store all the `Name` objects for a particular year. **Unlike in the first project**, this class should inherit from your own `AVLTree<Name>` class. (It should not have a data field of type `AVLTree<Name>` or `ArrayList<Name>`).

The class needs to provide the one parameter constructor

```
public YearNames ( int year )
```

[2] You may want to explore the methods of the `String` class that ignore the case when comparing two objects.
[3] A default constructor is one that can be used without passing any arguments.

Joanna Klukowska
joannakl@cs.nyu.edu

The constructor should throw an `IllegalArgumentException` if the `year` argument is not a four digit positive number between 1900 and 2018 (inclusive).

The class should implement

- `public void add ( Name name )`

  method that adds a new `Name` object to the list.

- `public int getYear ( )`

  method that returns the year number associated with this object.

- `public int getCountByName ( String name )`

  method that returns the number of babies that were given the `name` specified by the argument. This should include the names matching two different genders and all counties.

- `public double getFractionByName ( String name )`

  method that returns the fraction of babies that were given the `name` specified by the argument (this is the number of such babies divided by the total number of babies born in the year).

- `public int getCountByNameCounty ( String name, String county )`

  method that returns the number of babies that were given the `name` in the `county` specified by the argument. This should include the names matching both genders.

- `public double getFractionByNameCounty ( String name, String county )`

  method that returns the fraction of babies that were given the `name` in the `county` specified by the argument (this is the number of such babies divided by the total number of babies born in that county in the year).

This class should override the `equals` methods. The two `YearNames` objects should be considered equal if their years are identical.

The class should override the `toString` method. The details are up to you, but you should make sure that it returns a `String` object that is a meaningful representation of the object on which it is called (it may or may not contain the listing of all of the elements).

You may implement other methods, if you wish.

You will need to instantiate one `YearNames` object for each year in the dataset.

Efficiency Most of you observed that the `YearNames` class implemented in project 1 was fairly slow. Reading and storing the data from the full data set was causing a significant delay. In this project, the objective is to improve this performance by using a balanced binary search tree (an AVL tree).

Within the `YearNames` class there is additional room for making the implementation efficient. The two `getCountBy...` methods should be implemented in a way that does not visit every single node in the tree, but rather decides which branches of the tree should and should not be explored. The efficient tree exploration is discussed at the end of this document.

`AVLTree<E>` Class

This class should provide an implementation of an AVL tree. It should be based on the given `BST_Recursive` class, but certain methods in this class should be modified to implement the AVL tree rotations that guarantee that the tree stays balanced.

Your class should provide all of the public methods implemented in the `BST_Recursive` class and it should use identical signatures for those methods. Some of those methods will have the same (or almost the same) implementations, others will require significant rewriting and redesign. You may add other methods as you see fit. You will need to adjust the definition of the internal `Node<E>` class so it is applicable to the AVL tree implementation.

Joanna Klukowska
joannakl@cs.nyu.edu

`NYSBabyNames` Class

The `NYSBabyNames` class is the actual program. This is the class that should contain `main` method. It is responsible for opening and reading the data files, obtaining user input, performing some data validation and handing all errors that may occur (in particular, it should handle any exceptions thrown by your other classes and terminate gracefully, if need be, with a friendly error message presented to the user).

You may (and probably should) implement other methods in this class to modularize the design.

Given Code

To simplify parsing of the data from the input file, you can download from the course website the code for the following function:

`splitCSVLine` method allows you to parse the lines from the input file into an `ArrayList<String>` object containing all of the entries. This function produces empty strings to represent entries that were blank within the input line. This is the same function that you were using for project 1.

`BST_Recursive` class provides the recursive implementation for a binary search tree. This class does not need to be included in the final project. You should use it to guide you through the implementation of the `AVLTree<E>`. Feel free to copy/use any parts of this class that may be useful for the `AVLTree<E>` class. Keep in mind that you will need to rewrite the internal `Node<E>` class.

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at https://joannakl.github.io/cs102_s18/notes/CodeConventions.pdf.

The data files should be read only once! Your program needs to store the data in memory resident data structures.

You may not use any of the collection classes that were not covered in cs101 (for this assignment, do not use `LinkedList`, `Stack`, `Queue`. `ColorSet`, `PriorityQueue`. or any classes implementing `Map` interface).

You may use any exception-related classes.

You may use any classes to handle the file I/O, but probably the simplest ones are `File` and `Scanner` classes. You are responsible for knowing how to use the classes that you select.

Grading

If your program does not compile or if it crashes (almost) every time it is run, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

50 points class correctness: correct behavior of methods of the required classes and correct behavior of the program

15 points design and the implementation of the required classes and any additional classes

15 points efficient design

20 points proper documentation, program style and format of submission

How and What to Submit

For the purpose of grading, your project should be in the package called `project5`. This means that each of your submitted source code files should start with a line:

```
package project5;
```

Joanna Klukowska
joannakl@cs.nyu.edu

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to Gradescope.

You can produce a zip file directly from Eclipse:

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next
- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select "Save in zip format", "Compress the contents of the file" and "Create only selected directories"
- click Finish

Appendix

Sample interaction:

```
asia@zeppo$ java -cp bin project5.BabyNames ../csci102/projects/project5/Baby_Names_Beginning_2007.csv
Please enter a name:    Joanna
Please enter a county (ALL, for search in all counties):        all
2007 (0.0583): ||||||
2008 (0.0623): |||||||
2009 (0.0506): ||||||
2010 (0.0515): ||||||
2011 (0.0451): |||||
2012 (0.0596): ||||||
2013 (0.0401): |||||
2014 (0.0372): ||||
2015 (0.0320): ||||
Please enter a name:    Joanna
Please enter a county (ALL, for search in all counties):        Kings
2007 (0.1375): |||||||||||||
2008 (0.1036): ||||||||||
2009 (0.1038): ||||||||||
2010 (0.0992): |||||||||
2011 (0.1315): |||||||||||||
2012 (0.1114): ||||||||||
2013 (0.1037): ||||||||||
2014 (0.1056): ||||||||||
2015 (0.0387): ||||
Please enter a name:    Joanna
Please enter a county (ALL, for search in all counties):        Suffolk
2007 (0.0375): ||||
2008 (0.0783): ||||||||
2009 (0.0000):
2010 (0.0509): ||||||
2011 (0.0000):
2012 (0.0461): |||||
2013 (0.0000):
2014 (0.0000):
2015 (0.0316): ||||
Please enter a name:    q
```

```
Please enter a name:    Zoe
Please enter a county (ALL, for search in all counties):        New York
2007 (0.5636): ||||||||||||||||||||||||||||||||||||||||||||||||||||||||
2008 (0.5338): ||||||||||||||||||||||||||||||||||||||||||||||||||||
2009 (0.6424): ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
2010 (0.6426): ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
2011 (0.6263): ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
2012 (0.7057): ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
2013 (0.7084): |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
2014 (0.5040): |||||||||||||||||||||||||||||||||||||||||||||||||||
2015 (0.3583): ||||||||||||||||||||||||||||||||||||
Please enter a name:    Zoe
Please enter a county (ALL, for search in all counties):        Richmond
2007 (0.0000):
2008 (0.0000):
2009 (0.0000):
2010 (0.0000):
2011 (0.0000):
2012 (0.0000):
2013 (0.5211): |||||||||||||||||||||||||||||||||||||||||||||||||||||
2014 (0.3323): ||||||||||||||||||||||||||||||||||
2015 (0.1327): ||||||||||||||
Please enter a name:    Zoe
Please enter a county (ALL, for search in all counties):        Manhattan
No such name/county in the dataset.
Please enter a name:    q
```

```
Please enter a name:    milan
Please enter a county (ALL, for search in all counties):        all
2007 (0.0000):
2008 (0.0000):
2009 (0.0110): ||
2010 (0.0155): ||
2011 (0.0173): ||
2012 (0.0096): |
2013 (0.0321): ||||
2014 (0.0736): ||||||||
2015 (0.0459): |||||
Please enter a name:    milan
Please enter a county (ALL, for search in all counties):        westchester
2007 (0.0000):
2008 (0.0000):
2009 (0.0000):
2010 (0.0000):
2011 (0.0000):
2012 (0.0000):
2013 (0.0000):
2014 (0.1135): ||||||||||||
2015 (0.0880): |||||||||
Please enter a name:    james
Please enter a county (ALL, for search in all counties):        new york state
No such name/county in the dataset.
Please enter a name:    Jolie
Please enter a county (ALL, for search in all counties):        new york
2007 (0.0000):
2008 (0.1186): ||||||||||||
2009 (0.0000):
2010 (0.0000):
2011 (0.0000):
2012 (0.1142): ||||||||||||
2013 (0.0000):
2014 (0.0000):
2015 (0.0056): |
Please enter a name:    q
```

Joanna Klukowska
joannakl@cs.nyu.edu

Efficiency

Examples of efficient way of traversing the binary search tree when searching for objects matching some criteria.
Consider the tree in figure 1. In each node, assume that the letters and numbers indicate the different properties by the data stored in the nodes that are used for *sorting* the elements.

When searching for objects matching "F" in the first value and a range of numbers from 5-40 in the second value, an efficient implementation should visit only the nodes shaded in gray. At each node the algorithm should be able to make a decision if the search should continue on the left, on the right or in both subtrees.

Similarly, an efficient implementation should visit only the nodes shaded in blue when searching for objects whose first value matches "A" and whose second value is in the range of 0-100.

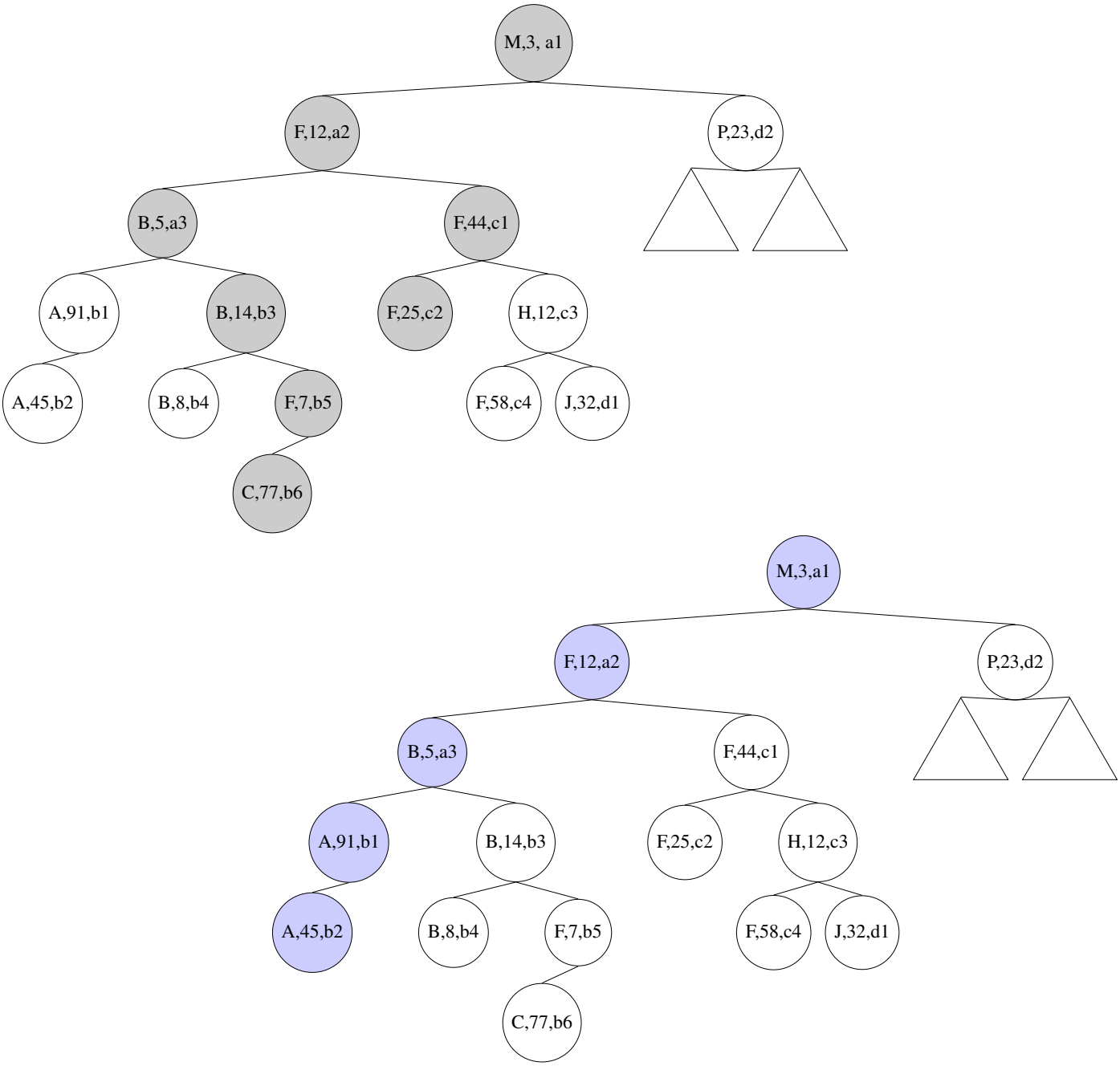

Figure 1: Efficient search for matching objects.