Joanna Klukowska
joannakl@cs.nyu.edu

Project 4: Baby Names: Improving Performance with Trees

Due date: November 21, 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 or any 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 your first project to modify its implementation in hope of improving its performance.

Some of the details in this project differ from the first project. Make sure you read the specification carefully and follow the requirements outlined in this project. The major parts that are different are indicated in a different color, but there may be minor changes in other sections as well.

You should be reusing as much of the code from the first project as possible. You should attempt to fix any problems and bugs that were discovered in your first project.

Your program will use the data provided by Social Security Administration that contains names given to babies born in US each year. Using this data and a name specified by the user, your program will need to generate a histogram showing the fraction of children who were given that name in each year.

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
- working with large data sets
- implementing and using binary search trees
- implementing generic classes
- working with existing code

Start early! You are reusing your existing code, but there is a significant part of the code that needs to be re-implemented.

Dataset

The data set that you need can be found at Data.gov[1] website that lists its download location at `https://www.ssa.gov/oact/babynames/names.zip`.

The file that you download is an archive zip file that contains files with data organized by year. There is also NationalReadMe.pdf file that contains detailed description of how the data is organized. In short, each file contains a long list of lines each in the format:

```
NAME,GENDER,COUNT
```

(see the NationalReadMe.pdf file for the description of format and restriction of each of the three fields).

[1] Data.gov is managed and hosted by the U.S. General Services Administration, Office of Citizen Services and Innovative Technologies. It hosts close to 200,000 datasets made available by federal, state and local governments as well as other organizations.

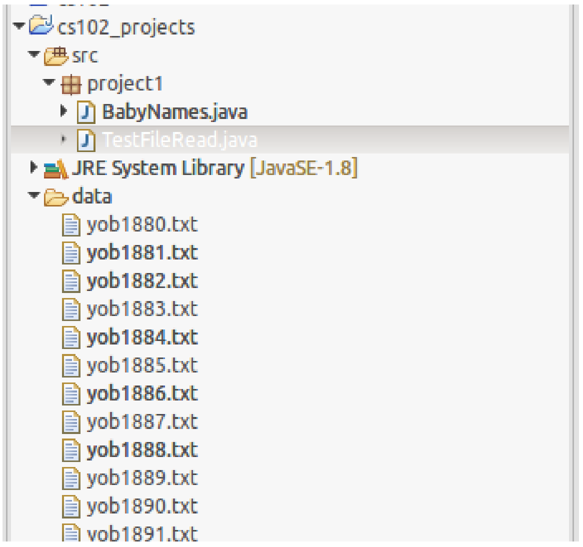
Joanna Klukowska
joannakl@cs.nyu.edu

Program's Data

For this program, the data files are considered to be part of the application (not the input provided by the user at runtime). You should store all of the data files in their own directory called `data` inside the project directory in Eclipse (this is at the same directory as `src` directory that stores the actual code, possibly in packages). This way you can open the files from within your program using the relative path, for example, `"data/yob1984.txt"`.

Within Eclipse the project/folder/package hierarchy should look something like the image on the right.

If you setup your program and data differently, the graders may not be able to run the code successfully.


```
cs102_projects
  src
    project1
      BabyNames.java
      TestFileRead.java
  JRE System Library [JavaSE-1.8]
  data
    yob1880.txt
    yob1881.txt
    yob1882.txt
    yob1883.txt
    yob1884.txt
    yob1885.txt
    yob1886.txt
    yob1887.txt
    yob1888.txt
    yob1889.txt
    yob1890.txt
    yob1891.txt
```

User Interface

Your program has to be a console based program (no graphical interface).

Program Usage

The program is started from the command line (or run within an IDE).

Unlike the first project, this program uses command line arguments. The two arguments specify the year range for the program to work with: the first argument indicates first year and the second argument indicates the last year. If either of the arguments is not a valid year number in the range from 1880 to 2015, or if the second argument is smaller than the first, the program should print an error message and terminate.

If the program is run with no command line arguments, then the default values of 1880 and 2015 should be used for the begin year and end year, respectively.

For example, running the program as

```
> java -cp bin proj4.BabyNames 2000 2015
```

should use the data files for years 2000 to 2015 (inclusive); running the program as

```
> java -cp bin proj4.BabyNames
```

should use the data files for years 1880 to 2015 (inclusive), but running the program as

```
> java -cp bin proj4.BabyNames 2015 2000
```

or

```
> java -cp bin proj4.BabyNames begin 2000
```

or

```
> java -cp bin proj4.BabyNames 2000
```

should produce an error.

Input and Output

The program should run in a loop that allows the user to check popularity of different names. On each iteration, the user should be prompted to enter either a name (for which the program computes the results) or the letter 'Q' or 'q' to indicate the termination of the program. If the user enters a name (i.e., the user does not want to quit) she/he should be prompted for the gender. (Unlike in the first project, the search in the project should be gender specific). The valid entries for gender should be single letter characters (following the rules of the data files) in either uppercase or lower case: 'F', 'f', 'M' or 'm'.

The user should not be prompted for any other response.

If the name/gender entered by the user cannot be found in the list of names stored in the dataset (i.e., it does not occur in ANY of the files), the program should print a message

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

and continue into the next iteration.

If the name entered by the user has matches in **at least one data file**, the program should print the histogram showing popularity of this name (see below for the details of formatting). The years that do not contain the name should be indicated by a "blank" histogram line (the value of 0.0 and no bars printed). Once the results are displayed, the program should continue into the next iteration.

Any error messages generated by the code should be written to the `System.err` stream (not the `System.out` stream).[2]

Histogram format:

If the name entered by the user is found in at least one data file, a histogram showing the popularity of this name should be displayed. If a name does not occur in all of the files, the program should set the number of babies who are given this name in a particular year to zero. For each years data, the program needs to determine the percentage of babies given a particular name

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

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[3]),
- `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

$$\left\lceil \frac{\text{number of babies with a given name}}{\text{total number of babies in that year}} \right\rceil \times 100$$

  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:

```
1880 (0.0129): ||
...
1984 (0.0960): ||||||||||
...
2015 (0.0287): |||
```

(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 data file, 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.

[2] If you are not sure what the difference is, research it or ask questions.
[3] You should be using `printf` for that. If you are not familiar with formatted output in Java, research it or ask questions.

- If the name/gender combination occurs in the data file more than once, it is an error and the corresponding line should be skipped. This means only the first occurrence of any name/gender should be used in the program.
- 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 of if the user types 'joanna', 'JOANNA', 'JoAnNa' or any other variation of cases.[4]
- Some names have several different spellings in common use, for example 'Joanna', 'Johana', 'Joannah'. For the purpose of this program these are considered to be completely different names.

Data Sorage 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.

`MyBST` Class

The `MyBST<E>` class is a container class that follows the binary search tree ideas that we are discussing in class. You need to provide your own implementation of this class. It can be viewed as somewhat simplified version of the `TreeSet<E>` class from Java API. Your class should implement a subset of methods provided by the `TreeSet<E>` class:

`MyBST()`
Constructs a new, empty tree, sorted according to the natural ordering of its elements.

`boolean add(E e)`
Adds the specified element to this set if it is not already present.

`boolean contains(Object o)`
Returns true if this set contains the specified element.

`E first()`
Returns the first (lowest) element currently in this set.

`boolean isEmpty()`
Returns true if this set contains no elements.

`Iterator<E> iterator()` Returns an iterator over the elements in this set in ascending order.

`E last()`
Returns the last (highest) element currently in this set.

`boolean remove(Object o)`
Removes the specified element from this set if it is present.

For detailed description of each of the above methods see the `TreeSet<E>` class documentation: `https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html` Your implementation is required to match those methods in terms of their behavior (types of parameters, return values, exceptions thrown and functionality).

You may implement additional methods, if you wish.

Note, that `MyBST<E>` is a generic class, i.e., it can store elements of any type as long as it implements the `Comparable<E>` interface.

This class provides general binary search tree. It should not be specific in any way to its use in this particular project in the context of BabyNames program.

NOTE: This class does NOT inherit from the 'TreeSet' class. You need to implement all of the above methods from scratch.

There are methods in this class you may not need to use in the program. You do need to implement them and test them even if you are not using them.

[4] You may want to explore the methods of the `String` class that ignore the case when comparing two objects.

`Node` Class

The `Node<E>` class provides the building blocks for `MyBST<E>` class. The generic type should be restricted to types that implement `Comparable<E>` interface.

This class should provide three constructors:

- the default constructor,
- the one parameter constructor that takes an element of type `E` as argument, and
- the three parameter constructor that takes an element of type `E` and two references of type `Node<E>` called `left` and `right`

The class should provide accessors and mutators for all data fields (data, left and right references).

The class should implement `Comparable<Node<E>>` interface.

The class should implement the `toString()` method.

`Name` Class

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

This class should provide a three parameter constructor:

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

If the constructor is called with an empty string for `name`, invalid `gender` indicator (valid values are single charactes 'f' for female, 'm' for male in either lower- or uppercase), or a negative value for `count`, then an instance of `IllegalArgumentException` should be thrown carrying an appropriate error message.

There should be no default constructor.[5]

This class should implement `Comparable<Name>` interface. The comparison should be done by the `name` as the primary key (using alphabetical order), by the `gender` as the secondary key and by the `count` as the tertiary key.

This class should override the `equals` methods. The two `Name` objects should be considered equal if the `name`, `count` 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.

The class should **inherit** from `MyBST<Name>` class (this means that the `YearNames` class is a binary search tree, but specific to the `Name` class).

In addition to the methods inherited from `MyBST<Name>` class, this class needs to provide its own methods that are particular to this project and to the type of elements that are stored in the tree.

The class needs to provide the one parameter constructor

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

The class should implement:

`public void add ( Name n )`

method that adds a new `Name` object to the list for a current year. This method should throw an instance of `IllegalArgumentException` if it is called with a `Name` object such that its name/gender data fields match an already existing object in this `YearNames` object. (This method should use the `add` method implemented in the `MyBST` class, but may need to decide in advance if the parameter object should be added or not.)

[5] A default constructor is one that can be used without passing any arguments.

Joanna Klukowska
joannakl@cs.nyu.edu

`public int getCountByName ( String name, String gender )`

method that returns the number of babies of a given `gender` that were given the `name` specified by the parameter. The `name` parameter can be any valid not-null `String` object. The `gender` parameter should be a one character string matching "F", "f", "M" or "m". If either of the parameters is invalid, the method should throw an `IllegalArgumentException`.

`public double getFractionByName ( String name, String gender )`

method that returns the fraction of babies of a given `gender` that were given the `name` specified by the parameter (this is the number of such babies divided by the total number of babies in the data file for the year). The `name` parameter can be any valid not-null `String` object. The `gender` parameter should be a one character string matching "F", "f", "M" or "m". If either of the parameters is invalid, the method should throw an `IllegalArgumentException`.

All of the above methods have to provide O(h) implementation where h is the height of the tree object - ideally $h \approx \log_2 n$ where n is the number of nodes in the tree. Since the `MyBST` class does not guarantee a balanced tree, the performance may be worse than O($\log_2 n$), but should always be O(h).

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 data set.

`BabyNames` Class

The `BabyNames` 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.

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at `http://cs.nyu.edu/~joannakl/cs102_f16/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 classes that were not covered in cs101 (for this assignment, do not use `LinkedList`, `Stack`, `Queue`. `TreeSet`, `PriorityQueue`. or any classes implementing `Map` interface).

You may use any exception-related classes.

Working on This Assignment

You should start right away!

You should modularize your design so that you can test it regularly. Make sure that at all times you have a working program. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

You should make sure that you are testing the program on much smaller data set for which you can determine the correct output manually. You can create just a few files that contain a few names each.

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

Joanna Klukowska
joannakl@cs.nyu.edu

You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens.

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.

10 points program correctness (the correct values and format of output) - note that if your program does not produce correct results, you will most likely loose points for the implementation part since the implementation itself is most likely incorrect.

60 points design and implementation of the four required classes and any additional classes

- `MyBST<E>` and `Node<E>` classes (30 points)
- `Name` class (5 points)
- `YearNames` class (20 points)
- `BabyNames` class (5 points)

10 points efficient design (this refers to efficient implementation of methods in the `MyBst` and `YearNames` classes as well as the processing of input files.

20 points proper documentation, program style and format of submission

How and What to Submit

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

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 our course materials or ...)
  - in Options select ”Save in zip format”, ”Compress the contents of the file” and ”Create only selected directories”
- click Finish

Do not submit the data files in your zip file!

If you wish to use your (one and only) freebie for this project (one week extension, no questions asked), then complete the form at `https://goo.gl/forms/b4IacXCvejIk5uOP2` **before the due date for the assignment**. All freebies are due seven days after the due date and should be submitted to NYU Classes.