**Summary**

Build two classes (Fraction and FractionCounter) and a Driver class for use in counting the number of unique fractions read from a text file (all fractions should be positive).   We’ll also reuse the ObjectList class we built in lab to store our list of unique FractionCounters**,** instead of directly using arrays or the ArrayList. Rather than designing a monolithic chunk of code in main like we did in the previous homework, we’ll practice distributing our code into containers (called classes) that you will design specifically to tackle this same problem in a different and more logically organized fashion.

**Introduction**

Your project is to read in a series of fractions from a text file, which will have each line formatted as follows: “A/B”.  **Both A and B are NOT NECESSARILY whole numbers.**  A sample text file is listed below, and the purpose of your program is to read in one line at a time and build a Fraction object from A and B.  For each unique Fraction seen, your program will create a FractionCounter object used to track the number of occurrences of that specific fraction.  When all the input is consumed, your program will print out its ObjectList of unique FractionCounters, which should report the fraction and its count – see output below. See the text file link ([fractions.txtPreview the document](https://canvas.uw.edu/courses/1331881/files/58017705/download?wrap=1)) for the some of the input I may use when testing your submission. Your program must reduce fractions, as demonstrated in the output below.

**Sample text file input:**

6/3

7/3

6/3

2/1

**Sample output:**

2/1 has a count of 3

7/3 has a count of 1

**Building Multiple Classes**

**Class Fraction**

This class should be a simple abstraction (i.e., a small class) that represents the ratio of two numbers.  There will be only two data elements, and only a few methods.  Note that **it is required for your Fraction class to store your ratio in reduced form, with the numerator and denominator as whole numbers,** so this is a feature to implement in your software.

**Data Members**

* The numerator – *what primitive type makes the most sense here?*
* The denominator – *same type as the numerator*

**Method Members**

* **Overloading**
  + Fraction() – default  “no-arg” constructor
  + Fraction(int,int) – constructor which initializes your data given the numerator and denominator of the fraction as entered in (which may be different than how the fraction is stored)
  + Fraction(double,double) – constructor which initializes your data for floating point versions of the numerator and denominator as entered in (which may be different than how the fraction is stored)
  + You may create additional constructors if you wish.
* **Overriding**
  + boolean equals(Fraction other) – compares “this” to “other”
  + String toString() – replace this inherited method with one appropriate to the class
* **Getters & Setters:** Note that the class does not include the traditional setters for numerator and denominator. This is because setting one at a time makes it impossible to guarantee the fraction is stored in reduced form. Instead, you will create a setFraction method that enables you to input the numerator and denominator at the same time (which may be different than how the fraction is stored):
  + getNumerator()
  + getDenominator()
  + setFraction(int,int)
  + setFraction(double, double)

**Class FractionCounter**

This class should also be small and contain only a few state variables.  The purpose of this class is to store a reference to a Fraction object and a count of how many times this fraction object has been seen in the input file.

**Data Members**

* The Fraction – declare this class variable to be of type Fraction from above
* The counter – the integer value used to count the number of these fractions seen

**Method Members**

* FractionCounter( Fraction theFraction ) – constructor
* bool  compareAndIncrement( Fraction newFraction ) – used to see if the newFraction passed into this function is the same as the Fraction we’re storing, and if so increments the counter by one and returns true (otherwise, returns false).
* String toString() – replace this inherited method with one that prints out both the Fraction and its count

**Testing and Boundary Cases**

Before submitting your assignment, be sure to thoroughly test it – for example, if your software fails to compile, the maximum grade drops substantially.  In addition to the earlier description as to what kind of output the program should generate, here is how the program should behave in light of different kinds of input:

* Valid, positive fractions should be correctly counted.
* Mathematically incorrect input for a positive fraction (e.g., 4/0, -1/4, 3.4/-2.3) should be skipped and ignored and the program continue to process the rest of the file.
* Mathematically correct input that yields a positive fraction should be handled correctly and kept (e.g., -1/-4, 4.2/3, 2.1/5.4).  Note that 2.5/1.25 and 6/3 have the same rational number value and would both contribute to the count of the fraction 2/1.
* Reasonable typos (e.g., blank lines) should be skipped and not cause the program to end.
* Unreasonable typos (e.g., those with alphabetic characters, multiple fractions on a single line, no divide sign) should cause the program to end because they indicate a corrupt file.
* Anytime the program ends, it has to be graceful (e.g., a friendly exit message with a system exit call).  Nothing should crash uncontrollably.

Based on the above description, here are some cases you should test.  You are not only testing your code works with expected cases but also boundary cases or unusual situations.  This list is not exhaustive but is a first cut at what to test for:

* Does your software successfully count valid, positive fractions?    Test this common case works.
* Does your software reduce fractions, including fractions where the numerator and denominator (as entered in) are decimal numbers?  Test this common case works.
* Does your software read in the fraction "4/0" and skip and ignore it?
* Can your software handle input fractions made of floating point numbers?
* Can your software handle a file with no fractions? 1 fraction?  1,000 fractions?  Blank lines between fractions?
* What does your software do with the fraction "-1/-4"?  "-1/4"?  "efw3"?  "1/3 5/3"?  "321"?  As described above, your program should handle these differently.
* What if the numerator or denominator is very large, as in "1/9999"?
* Does your program report the fraction "1/1 occurs 4 times" only once (as it should, similar to Fractions V1), or does it report "1/1 occurs 4 times" multiple times (which it should not)?
* If you store the fraction using doubles, does the equals method suffer from roundoff errors?

Test your code for as many cases as you can think of by varying the *fractions.txt* input file.  Note that we may try multiple *fractions.txt* input files when grading your assignment.

NB:  You **need to code your program it so it can run with input files besides the example one.** In particular, it has to work with input files that are of any length.  **You must enable the appropriate lists (such as ObjectList) to dynamically resize and must accomplish this without using the resizing capabilities built-in to some standard Java classes (e.g., ArrayList).**

**Notes and More Requirements**

* Test your code as you add features (either methods or data) incrementally, making sure your small change works as you envisioned it to.
* Your code should correctly specify the input file as *fractions.txt*, **without** an absolute path or any other environment-specific specifications.  The program should **not** ask for user input from the console.
* Try to keep in mind OO principles; for example, to promote information hiding, we should probably make all of our data members private for each class.
* Consider reading one line at a time (with the Scanner nextLine() method), and using the [split (Links to an external site.)](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html#split(java.lang.String)) method (defined in class String) to give you an array with two elements in it (the numerator and denominator, respectively)
* Consider adding a few helper methods to your ObjectList class, such as indexOf(), contains(), get(), and toString().  Note the get() method probably should return a FractionCounter object; if you do this, you have to explicitly typecast into a FractionCounter object, i.e., with a "(FractionCounter)a" token, where "a" is the element from the ObjectList array.
* The Object class enables you to write parameter lists in Java that accepts arguments of any class. Thus, the ObjectList class will ultimately create and manage a list of any kind of object. In lab, you managed different shapes, like Square and Circle. In this homework, you will manage a list of FractionCounter objects with an instance of the ObjectList class (see below). (This will make more sense when we discuss inheritance, but if you're interested in a reading on this, see Savitch pp. 464-465.)
* Getting a null pointer exception or all Fractions seem to have the same value?  Building an array does not put any elements inside the array, so make sure you are building new Fraction objects and adding them to your list if they are unique.
* You should use build a FractionCounter object for each unique Fraction as your program executes.
* When reporting each unique fraction, it needs to be in reduced form (Euclid's GCD theorem will help you here).  **By "unique fraction" I do not mean "unique string value" but "unique rational number value."**
* You can determine if a Fraction is unique by looping over your list of FractionCounters, calling compareAndIncrement() on each FractionCounter, passing as input the Fraction you just read from the file.  If all calls to compareAndIncrement() return false, then this is a unique Fraction that needs a FractionCounter created for it.
* The FractionCounter function compareAndIncrement() will have to compare two fractions, and this will require some kind of comparison function for the Fraction class.  You should implement some type of “equals” method that simply compares the numerators and denominators of the two Fractions in question. One approach would be to define a static “isEqual” function that takes two Fraction objects as input, and gets both numerators and both denominators.  If they are both equal, then the function returns true, otherwise false.
* **Comment your code.** In particular, provide Javadoc comments at the top of the file (describing the class) and before each method (including main).  In the method comments, describe what goes in/out of the method and what the method does.
* Classes whose names are given in this assignment description should be given that name.  If you submit multiple versions, Canvas will rename your files with a "-1", etc. suffix, but that's okay.  We can handle that change.  But the base name of your class has to be as described.

**What to Submit**

Note that you should **not** put your name as part of the filenames for the classes you turn in.

* The Fraction class
* The FractionCounter class
* The ObjectList class
* A Driver class that just has a driver in its main

Please **submit your files inside a zip archive called FractionsV2.zip.**  If you are not sure how to create a zip archive, do a Google search on "create zip archive windows" (swap out your computer's operating system, as needed).  Make sure you give yourself enough time before the due time to learn about and create the archive!

**About This Document**

Original assignment by Rob Nash, Autumn 2014. Last minor edits and additions by Johnny Lin, August 2019.

**Rubric**

Fraction V2 Rubric (2)

| Fraction V2 Rubric (2) | | |
| --- | --- | --- |
| **Criteria** | **Ratings** | **Pts** |
| This criterion is linked to a Learning Outcome Comments in Code |  | 5.0 pts |
| This criterion is linked to a Learning Outcome Code Executes |  | 5.0 pts |
| This criterion is linked to a Learning Outcome Code Runs & Accomplishes Outcomes |  | 5.0 pts |
| This criterion is linked to a Learning Outcome Use of Functions |  | 5.0 pts |
| This criterion is linked to a Learning Outcome Built Required Classes |  | 7.0 pts |
| This criterion is linked to a Learning Outcome GCD reduction |  | 3.0 pts |
| Total Points: 30.0 | | | |

**Submission**

Submitted!

Oct 10 at 11:26pm

[Submission Details](https://canvas.uw.edu/courses/1331881/assignments/4923146/submissions/3729452)

[Download FractionsV2.zip](https://canvas.uw.edu/courses/1331881/assignments/4923146/submissions/3729452?download=58872315)

Grade: 25 (30 pts possible)

Graded Anonymously: no

[View Rubric Evaluation](https://canvas.uw.edu/courses/1331881/assignments/4923146/submissions/3729452#rubric)

**Comments:**

Your program crashes if fractions.txt contains a string. It also uses parseInt(), which means that it will crash for all doubles. Your ObjectList should be able to expand to any number of elements. Limiting the array size to 150 Fractions caused your program to crash when I attempted to count 10,000 fractions.