Fractions V2

Re-submit Assignment

Due Oct 14 by 11:59pm **Points** 30 **Submitting** a file upload **File Types** zip

Summary

Build two classes (**Fraction** and **FractionCounter**) and a **Driver** for use in counting the number of unique fractions read from a text file. 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.

Remember

- NO DECIMALS!
- Handle input of any length

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". 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. You can assume no blank lines or misleading characters; see the text file link above for the some of the input I'll 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, so this is 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
- **Overriding**
  - boolean equals(Fraction other) – compares “this” to “other”
  - String toString() – replace this inherited method with one appropriate to the class
- **Getters & Setters**
  - getNumerator()
  - setNumerator(…)
  - getDenominator()
  - setDenominator(…)

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

Error handling

- When encountering illegal data, print an error message, skip the data, and continue processing

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 falls to a 75%. Does your software reduce fractions? Test this common case. What about boundary cases or unusual situations? For example, does your software read in the fraction “4/0” and crash with a DivideByZeroException?   Can your software handle 0 fractions? 1 fraction? 1,000 fractions? What does your software do with the fraction “-1/-4”? 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, for the fractions.txt file linked in 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() function 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 I will 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 to dynamically resize.**

Hints (& Spoilers)

- Test your code as you add features (either methods or data) incrementally, making sure your small change works as you envisioned it to.
- 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 (http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html#split(java.lang.String))** function (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).
- You can determine if a **Fraction** is unique by looping over your list of **FractionCounter**s, 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 **Fraction**s 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.

What to Submit

Submit a single .zip file with the following files inside it. If you are not familiar with creating .zip files, see the document Creating zip files. 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

About This Document

Original assignment by Rob Nash, Autumn 2014. Minor edits and additions by Johnny Lin, January 2015 and Lesley Kalmin January 2019

Fraction V2 Rubric (1)

Criteria	Ratings				Pts
Comments in Code Javadoc style for each File, class, method	**5.0 pts** **Excellent**	**3.0 pts** **Good**	**2.0 pts** **Average**	**0.0 pts** **Few/None**	5.0 pts
Code Executes	**5.0 pts** **Full Marks**	**0.0 pts** **No Marks**			5.0 pts
Code Runs & Accomplishes Outcomes	**5.0 pts** **Full Marks**	**3.0 pts** **Partial Marks**	**0.0 pts** **No Marks**		5.0 pts
Handle input cases Zero denominator, negative values, resize array	**4.0 pts** **Full Marks**	**2.0 pts** **Partial Marks**	**0.0 pts** **No Marks**		4.0 pts
Built Required Classes Define an d use methods given	**8.0 pts** **Full Marks**	**4.0 pts** **Partial Marks**	**0.0 pts** **No Marks**		8.0 pts
GCD reduction Include source citation	**3.0 pts** **Full Marks**	**2.0 pts** **Partial Marks**	**0.0 pts** **No Marks**		3.0 pts
					Total Points: 30.0