Computing and Algorithms I
Project 4

CS-101 **Summer, 2013**

Project 4 is a 100 point assignment in which you will be able to practice problem decomposition, top down design, and top down implementation. For this project we will be dealing with fractions. You will also be doing file input and output, where the file names will be command line arguments. Note: all your input will come from a file; all your output will go to a file. The names of both of these files will be command line arguments to your program.

The Problem: Arithmetic on Fractions

You will solve the problem of reading arithmetic problems involving fractions from a multiple line file, finding the results of doing the arithmetic, and printing the results to a file. A fraction will be defined as n/d, where n and d are integers, n is the numerator and d is the denominator. For example, $7/10$ and $-8/-4$ would be examples of fractions for input.

Format of the Input

The operations you will be doing in your code will be:

- add – use the string add (capitalization does not matter)
- subtract – use the string sub (capitalization does not matter)
- multiply – use the string mul (capitalization does not matter)
- divide – use the string div (capitalization does not matter)
- reciprocal – use the string rec (capitalization does not matter)

Your input will be from a file, the last line of the file processed will contain the word quit in any form of capitalization. The line containing the word quit may contain any additional characters anywhere in the line. The line containing the word quit is not used in any way other than stopping the input for the program. The lines preceding the quit line will contain one arithmetic problem per line of input. An input line will take one of the following forms:

- a list of fractions separated by operators add, mul, sub, and/or div. For example

  $5/-10$ add $4/2$ sUb $652/1337$ DiV $-2/435$ MUL $-7/-14$

  White space is allowed before the first fraction and after the last fraction, but the operators will be surrounded by white space.

- The operator rec followed by one fraction. For example

  rEC 7/19

  White space is allowed before the operator and after the fraction, but there will be white space separating the reciprocal operator from the fraction.

Your last input line will contain the word "quit", where again capitalization does not matter, and there can be any characters at all before or after the word "quit".

Form of the fraction

You will write a Fraction class which will have the methods to add, subtract, multiply, divide, and take reciprocals of fractions. This class will also have a toString method and at least one constructor. There will be no print statements anywhere in the Fraction class.

Your Fraction class will store the fraction in reduced form in a Fraction object. There will be an instance variable for the denominator and for the numerator. For example, if the fraction is 2/4 your Fraction class will store the Fraction object data as 1/2, with the 1 and the 2 stored in the appropriate instance variables. To find the greatest common divisor, treat both the numerator and denominator as positive integers and find the largest positive integer which evenly divides both. (The greatest common divisor code should be in its own method).

The denominator will never be shown as negative, you will store the sign with the numerator. For example, if you create a Fraction with $3/-5$, your code will store the Fraction as $-3/5$, and this is the way it will be displayed.

A denominator can never be 0. If you somehow end up with a 0 denominator, your code will print the word "undefined" for the value of the Fraction.

Output

If the input line is a list of fractions separated by operators, you will perform the operations from left to right showing the partial results at each step. For example, if the following is an input line:

```
    5/10 add 1/3 mul 4/6
```

Your output should have the form

```
    5/10 add 1/3 mul 4/6
       1/2
       add 1/3 equals 5/6
       multiply by 2/3 equals 5/9
```

Note that the above output echoes the line of input, then performs any necessary reduction to the Fraction before printing it on its own line, then for each operation, specify the operation and do it, showing the result.

If the input is the reciprocal operator, you will show the reciprocal in the following form. If the input is:

```
    rec −2/5
```

Your output will be

```
    rec −2/5
        the reciprocal of −2/5 is −5/2
```

If at any point you have an undefined fraction, all operations on that fraction will again result in an undefined fraction. Your output will show this.For example, if the following is an input line:

```
    5/0 add 1/3 mul 4/6
```

Your output should have the form

```
    5/0 add 1/3 mul 4/6
        undefined
        add 1/3 equals undefined
        multiply by 2/3 equals undefined
```

Solution Process

You will solve this problem using top down design and implementation. To show that you are doing this, you will have at multiple versions of the solution, where each version builds on the previous version. You will show three of these versions as part of the package you create to solve this problem. Your code will grow in each version similar to the implementation of the number base conversion problem we designed in class. Each of your versions of code should be in a separate folder (file directory). Your first version will have only the main method completed, all other methods and classes in that version will be stubs. The code in main in version 1 should not change at all for later versions.

Deliverables

On Monday August 26 you will turn in Version 1 which is described below. You will zip all java source files into a single file with the name project4V1.zip and submit this file via blackboard before the beginning of class. You will also turn in a printout of all files described below for Version1.

On Thursday August 29 at the beginning of class all the rest of the project is due.

Version 1

Version 1 will do no more than read each input line of the file, print the line to the output file, and stop when the line containing the word quit is read from the input file. Capitalization of the word “quit” does not matter. There should be a call to one or more methods which will be implemented as stubs to solve the rest of the problem. (Note that your main method from Version 1 should not change in later versions of your problem solution). You will turn in a UML class diagram for your class containing the method main, plus a UML class diagram for all classes containing stubs called from main. You will have a data table for each class that has a constructor. You will have a data table for the main method. You will have an algorithm for main. These design documents will be created using a spread sheet program. You will also include algorithms and data tables as comments in the java code appropriately.

Remainder of Project Solution

On Thursday August 29 at the beginning of class you will turn in a document printed from a printer containing the following items:

1. A UML design containing UML class diagrams and a UML class interaction diagram showing all classes you wrote for your solution. A data table for each class you wrote. Algorithms and data tables for each method you have created. Exceptions: you do not need to write an algorithm or a data table for any accessor methods.

   Note, you are doing this for both versions you turn in on Thursday.

   Note, you are turning in the final version, and a version built on the first version which is due on Monday of week 7, which you will build on to create the final version.

2. The java code for your version which is not the final version.

3. The java code for your final version.

4. The input file and output file from running your final version. Your code should print the problem title, your name, and the course before processing the input.

Be sure to staple all the papers together into one package, and that your name is printed (by the printer) on each of the separate documents, including the output. You can put your name in the input file by placing it after the quit line (your program will stop processing input at that point).

Submit your code to me on via blackboard by zipping these 2 versions of Java code together using the name project4final.zip (note that you can zip entire folders, so if you have your version code in separate subdirectories of a directory, you can zip all the versions at one time.

The blackboard submission of code and the printouts are due at the beginning of class.

Be prepared to demonstrate that your program works during class on Thursday August 29.