CS 3230 Lab #2
Classes, Regular Expressions, and Problem Solving

A fraction is composed of two integers: a numerator and a denominator, and supports (among others) the common four arithmetic operations: addition, subtraction, multiplication, and division.

Requirements

Lab #2 consists of two public classes named Fraction and FractionCalc.

Class Fraction defines two instance fields and must define the methods described below.

Instance fields in both classes must be private.

Class Fraction

Let F1, F2, and F3 be Fraction objects and let I be an integer (i.e., an int).

1. Two constructors:
   a. Fraction(int numerator)
      i. Makes a fraction from an integer
         ii. Set the denominator = 1
   b. Fraction(int numerator, int denominator)
      i. Reduce the fraction to lowest terms (see “Euclid’s Algorithm” below)
         ii. Improper fractions are okay
            iii. Improper fractions and fractions with a denominator of 1 are okay

2. Methods implementing the four arithmetic operators (use the method names illustrated - the class could be used in many other contexts):
   a. F3 = F1.add(F2);
   b. F3 = F1.sub(F2);
   c. F3 = F1.mult(F2);
   d. F3 = F1.div(F2);

3. Tests two Fraction objects for equality (i.e., override the equals method - follow the “recipe” given in Horstmann and Cornell - be sure to read the caution)
4. Override the toString method. Should produce output similar to: 5/4

Euclid’s Algorithm

Euclid’s algorithm may be used to find the greatest common divisor (i.e., the largest integer that evenly divides two other integers – e.g., the numerator and denominator):

```java
int divisor = gcd(numerator, denominator);
```

This should make reducing a fraction to lowest terms quite easy. (Why is gcd private?)

```java
// Euclid's Algorithm for finding the greatest common divisor
```
private int gcd(int u, int v)
{
    u = (u < 0) ? -u : u; // make u non-negative
    v = (v < 0) ? -v : v; // make v non-negative

    while (u > 0)
    {
        if (u < v)
        {
            int t = u; // swap u and v
            u = v;
            v = t;
        }

        u -= v;
    }

    return v; // the GCD of u and v
}

Class FractionCalc

Class FractionCalc implements a simple fraction calculator, which operates as follows:
1. Prompts the user to enter a fraction expression
2. Reads the expression as a string (the test bed sends the input through System.in)
3. Validates the expression using matches defined in the String class (see RegExDemo.java included with this assignment for a regular expression and an example of the matches method)
4. Parses the expression into a token stream
5. Converts the appropriate tokens into Fractions
6. Carries out the specified fraction operation
7. Prints the result
8. Ends (i.e., the program does not loop)

The basic input pattern is: [-]n1 / [-]d1 op [-]n2 / [-]d2

• where n1, d1, n2, and d2 are the numerators and denominators, which may optionally be preceded by a - sign (no space between the - and the number)
• the numerator and denominator are any sequence of the digits 0-9. The sequence length is greater than or equal to 1 and may be just the digit 0 (which will cause a problem that we ignore for now but which can be addressed with a simple modification to the provided regular expression)
• op is exactly one of +, -, /, or *
• each of the tokens above may be separated by 0 or more spaces or horizontal tabs
• the regular expression does not handle leading or trailing spaces in the input string; see the trim method in the String class
The package java.util.regex contains a class named Pattern. The documentation for this class explains the regular expression language used in RegExDemo.java. Can you see how the provided regular expression implements the above description?

Two ways of parsing the fraction expression are suggested below. Choose and complete one of these implementations or approach the problem in a different way if you wish.

**StringTokenizer**

java.util includes a class named StringTokenizer: “The string tokenizer class allows an application to break a string into tokens.” Read the details of the last two constructors carefully. The hardest part of this approach is dealing with the optional ‘-’ signs. I found it convenient to use an ArrayList<String> (also in the java.util package) with the StringTokenizer. See the get method, the charAt method, both add methods, and the remove method.

**Pattern and Matcher**

Both the Pattern and Matcher classes are defined in the java.util.regex package. (Note that if you import java.util.*, that statement allows you access all of the classes defined in the util package but does not allow you to access any subpackages - you must import java.util.regex separately.) This technique is more complex overall than the previous technique but quite interesting. To use these to classes to parse the input, you must

1. create patterns or regular expressions that match fragments or subexpressions in the input
2. build a Pattern object
3. make a matcher that matches the input in the pattern
4. find the pattern in the input
5. extract the group that matches the pattern

For example:

```java
Matcher f = Pattern.compile(fracPattern).matcher(expression);
if (f.find())
{
    String fraction = f.group();
}
```

fracPattern is a pattern or r.e. that matches a fraction: [-]n / [-]d (expressed in the language shown in the RegExDemo example). Note that find works left to right, if you use two matchers, the second may need to start somewhere inside the string (see end and both find methods in the Matcher class). You might also find the split and charAt methods in the String class useful.

**Grading**

Your program should handle all of the valid tests shown in RegExDemo.java. Upload your two .java files to Blackboard. Please do not zip the files, but do make sure that the files/classes are named correctly, and please make sure that you have removed all package statements.