HW01: Fractions Class
CS5004

Write an Interface, Fraction, to specify the protocol (methods) of a concrete class, FractionImpl. Although this Interface could then be implemented in several ways, you should then provide just one concrete implementation. An instance of FractionImpl represents a particular fraction with an integer numerator and a positive integer denominator. Your Interface should provide Javadocs specifying the semantics of each method. (Recall that the Interface does not include a *constructor*, nor does it specify *toString*; however, your Fraction implementation should override the default toString, as described below.) Your solution should include:

A. The Interface definition, including method signatures and Javadocs.

B. A **constructor** that takes a numerator and denominator as integers. If the fraction is negative, then the numerator should be negative. The denominator should always be positive. Throw IllegalArgumentException when appropriate.

C. **Getters** and **setters** for the numerator and denominator. Use care to ensure that a setter does not invalidate the Class's requirement that denominators must be positive.

D. A method **toDouble** that returns the scientific value (decimal) of the fraction. If needed, use (double)x to cast an int as a double.

E. A method **toString** that returns a String depicting the fraction's value, as a fraction, but in simplest form. For example, "4 / 2" should be simplified to "2 / 1" etc. Here is a simple, recursive version of Euclid's algorithm for finding the greatest common divisor of two integers; you may find it helpful:

```
static int gcd(int a, int b)
{
  if(b == 0)
  {
    return a;
  }
  return gcd(b, a % b);
}
```

F. A method **reciprocal**() that returns the reciprocal of this fraction. Beware of the case where the numerator of the original fraction is 0, and handle negative values with care.

G. A method **add**(Fraction other) that adds this fraction to the one passed to it and returns the result as a fraction. Consider that one or both fractions might be negative!

H. A method **compareTo**(Fraction other) that compares two fractions. It returns a negative integer if (this < other), a positive integer if (this > other) and 0 otherwise. Hint: cross-multiplication may be helpful. Note: this method implements *Comparable*<Fraction>.

I. Write JUnit tests for your class. Consider every use case you can anticipate and ensure that there is a test for correct behavior in each case. If an exception is anticipated, you should ensure that it will be raised if and only if appropriate.

J. Add sufficient Javadocs comments to make your implementation as well as your tests understandable to target users. Where appropriate, also include in-line comments that would

be helpful to another software engineer later assigned to maintain or extend your code. It is not necessary to merely repeat Javadocs that appear in the interface; but for example, your constructor and toString method should both have Javadocs comments.

The rubric for scoring will consider:

- completeness of solution
- correctness of solution
- adherence to (Google) style guidelines
- other style considerations such as choice of identifier names or duplicative code
- presence and quality of Javadocs where appropriate
- presence and quality of inline code commentary where appropriate
- completeness/correctness of tests for all methods.

Submit your solution on GitHub and the paste the URL to the correct folder into Canvas. Please do not forget the second step; timely submission involves both.