Chp 8 Programming Project

In this exercise you will extend your work on classes and implement a class that *encapsulates* the state and behavior of an improper fraction. The name of the class *must* be Fraction (in a file named Fraction.java). The class has the following requirements:

## REQUIREMENTS

* Two int fields representing a numerator and a denominator. The fields *must* be properly encapsulated to prevent direct access from outside the class. That means the fields *must* be private and you *must* implement accessor methods as appropriate.

You may not have any other fields.

The class must enforce the invariant that the numerator and denominator are fully reduced, and that the denominator is not negative.

* Three constructors that accept the sets of parameters described below and appropriately set the two fields described above.
  + The first constructor *must* accept two int parameters representing a numerator and denominator (in that order). Any two ints are valid, except the constructor *must* ensure that the denominator is not 0. Otherwise the constructor *must* throw an IllegalArgumentException similar to the following code

**if** (<condition>) {

**throw** **new** IllegalArgumentException("Message goes here");

}

* + The second constructor *must* accept a single integer parameter representing a wholeNumber. Any int is a valid whole number.
  + The third, and last, constructor *must* accept a single String parameter representing an improper fraction.
* An improper fraction has an optional leading sign character (+ or -), followed by a number, followed by a slash character, followed by a nonzero number. There MUST be no space between any of the characters. The general forms for an improper fraction are:

<improper> ::= <sign><number>/<number>

Valid examples for an improper fraction are:

+1/2

-2/4

0/7

For the purposes of this assignment, the input string is guaranteed to be valid.

* In order to maintain the invariant, all constructors *must* reduce the fraction. That is, if the user calls new Fraction(6, -8) , then the stored values for the numerator and denominator are -3 and 4 respectively.
* You must implement four instance (not static) methods that implement familiar mathematical operations. The methods *must* be named add, subtract, multiplyand divide*.*  Each method *must* take a single Fraction parameter and return a Fraction result. The methods must not modify the current object, or the parameter. In the case where the parameter represents the fraction 0, the divide method must throw an IllegalArgumentException
* Implement the toString() method for the Fraction class so that it returns the minimal string. That is, any value with no fractional portion *must* be returned as an appropriately signed whole number (e.g. -1, 7). Any other value *must* be returned as a reduced fraction with just a numerator and denominator (e.g. -1/2, 7/2).
* Implement the toMixedNumberString() method for the Fraction class. This method is identical to the toString() method for whole numbers, and for fractional numbers between -1 and 1. Fractional numbers smaller than -1 or greater than 1 must be displayed as a mixed number. A mixed number consists of an optional minus sign, followed by a whole number, followed by an underscore, followed by the numerator, followed by a /, followed by a denominator. For example, the improper fraction -5/4 corresponds to the mixed number -1\_1/4
* Implement the compareTo method. The method must take a single Fraction parameter and return an int. The int must follow standard compareTo patterns; it *must* be negative if the current Fraction is less than the parameter, positive if the current Fraction is greater than the parameter, and zero if the two Fractions are equal.

Your class *must* not display anything (with a print, println, etc.), and your class *must not* have a main method. You can then use the provided Test.java file (which has a main method) to test and verify the correctness of your implementation.

## Sample Usage

In this sample we declare and instantiate a Fraction f. We set the value to 6/4, then call println to display the values returned by f.getNumerator() and f.getDenominator(). This *must* display the values 3 and 2 respectively. Finally we multiply f by itself and print out the result – yielding the final value of 9/4.

Fraction f = new Fraction("6/4");

System.out.println(f.getNumerator());

System.out.println(f.getDenominator());

System.out.println(f.multiply(f));

## Scoring

1 pt (total of 2) each for proper field declaration and initialization

2 pts (total of 6) for each working constructor.

3 pts for proper reduction during construction

2 pts (total of 8) each for each working operator

3 pts for proper output from the toString() method

3 pts for proper output from the toMixedNumberString() method

2 pts for proper output from the compareTo() method

3 pts for proper coding style, indentation, whitespace, comments

Total of 30

## Extra Credit

In addition to the Fraction class, create a BigFraction class. The BigFraction class should be mostly identical to the Fraction class, except it can handle numerators and denominators with many more digits than the Fraction class. The BigFraction class will do this by using the java.math.BigInteger class for its numerator and denominator. BigFraction should keep the three constructors of Fraction, and in addition have two new constructors that take BigIntegers instead of ints. All methods that take a Fraction as a parameter should take a BigFraction instead.

You are expected to look up information about the BigInteger class, and figure out how to use it yourself. Discovering and using code you didn’t write by doing research and reading documentation is a big part of real life software development.