HW 8: Rational Number Object

Finish the `Rational` object that has been started, that represents a “rational number”.

A rational number is simply the ratio of two integers. They can be useful sometimes, since doubles have inherent rounding errors and are incapable of representing a number like $^1/_3$ with true accuracy.

There are many stub functions waiting for you inside of `Rational.java`, that don’t do anything yet. Your job is to fill them with code to make a working `Rational` object. Your object should follow these rules:

1. After every change to the numerator or denominator, they should immediately be reduced to lowest terms. So if you make a new `Rational` to hold $^4/_6$, it will immediately reduce itself down to $^2/_3$.
2. If the overall number is negative, the numerator should be negative while the denominator is positive. The denominator must *never* be negative.
3. It is legal if both numbers are 0: this is the equivalent of the double value NaN.
4. If the numerator is 0, the denominator needs to be 0 or 1. This is equivalent to NaN or 0.0, respectively.
5. If the denominator is 0, the numerator needs to be 0 or ± 1. This is equivalent to NaN or $\pm\infty$, respectively.

These are the public methods you need to implement:

- `add()`, which takes a second `Rational`, adds it to the first, and returns the sum.
- `subtract()`, which takes a second `Rational`, subtracts it from the first, and returns the difference.
- `multiply()`, which takes a second `Rational`, multiplies it by the first, and returns the product.
- `divide()`, which takes a second `Rational`, divides the first by it, and returns the quotient.
- `toDouble()`, which returns the double best approximating the `Rational`’s value.
- `toString()`, which returns the string `(numerator/denominator)` (with `numerator` and `denominator` replaced by the proper values).
- `equals()`, which takes a second `Rational`, and returns true if it’s equal to the first.
- `getNumerator()`, which is an accessor for the numerator.
- `getDenominator()`, which is an accessor for the denominator.
- `setNumerator()`, which is a mutator for the numerator.
- `setDenominator()`, which is a mutator for the denominator.

In addition, there are two private functions. Their implementation is not strictly necessary, but they will make your life easier.

- `reduce()` simply applies the above five rules to the fields. It is a void function, because it only affects the fields.
- `calcGCD()` calculates the Greatest Common Divisor of 2 ints. It is a static function, because it doesn't actually modify anything in the fraction. It just takes its input and returns an output, regardless of anything else.

The easiest way to calculate the GCD of two numbers a and b is this:

```
1. If one of the numbers is 0, just return 0.
2. Reorder them and take absolute values, so both numbers are positive and $a$ is larger.
3. Loop forever:
    • If ($a$ mod $b$) is 0, return $b$.
    • Otherwise, set $a$ to be the old $b$, and $b$ to be ($a$ mod $b$).
```

The final output should look like this:

```
First rational is: (1/2) (equivalent to 0.5)
Second rational is: (2/3) (equivalent to 0.6666666666666666)

Sum: (7/6)
Difference: (-1/6)
Product: (1/3)
Quotient: (3/4)

Are the rationals equal? false
Is the first equal to 1/2? true

The numerator of the first rational is 1.
The denominator of the first rational is 2.
Changing numerator to 6, and denominator to 7...
...and the result is:  (3/7)

Zero as a rational is (0/1) (equivalent to 0.0).
Negative infinity as a rational is (-1/0) (equivalent to -Infinity).
NaN as a rational is (0/0) (equivalent to NaN).
```

You can also test your `Rational` object using BlueJ's built-in capabilities, as you may have done for this week's lab.

Try to plan out the order in which you will work on these methods, and do them one at a time. Don't move on to another until you are *positive* that the current one works 100% of the time. Otherwise, when you hit a bug it can be very hard to know where the actual mistake is.

You just need to turn in `Rational.java`.