Rational Numbers Lab

A *rational number* is a number that can be expressed as the quotient of two integers, where the denominator is not zero. For instance, 1/2, 3/4, 40/17, and 5/1 are all rational numbers. C++ does not have a rational data type, and therefore stores all rational numbers as floats or doubles. This can cause problems because some rational numbers, such as 1/3, cannot be represented exactly in decimal notation (at least not with a finite number of digits). In this lab, you will create a simple `rational` class to better represent (positive) rational numbers.

1. Create a class called `rational`. This class should have two private integer fields, the numerator and the denominator. Add a getter for the numerator and one for the denominator. We will not add setters; our rational numbers will be *immutable* once constructed (like Python's strings).

2. Add two constructors to your class. One should be a "default" constructor (no arguments) that constructs the rational number 1/1. The other should be a constructor that lets the user construct any rational number they want. This constructor will take two integer arguments: a numerator and a denominator. You should never let the user create a rational number with a denominator of zero (call `exit(1)` if they try).

   Example of usage:
   ```
   rational one;            // construct the rational number 1/1
   rational onehalf(1, 2);  // construct a rational number representing 1/2
   ```

3. Add a method called `print()` to your class. This method will use `cout` to print the rational number to the screen. For example, `onehalf.print()` should print `1/2` to the screen (assuming you have the `onehalf` declaration line from question 2).

4. Rational numbers are usually always given in lowest terms (where the numerator and denominator have no factors in common other than 1). Add a method called `reduce()` that reduces a rational number to lowest terms. Should this method be public or private and why? Where should it be used if you want to guarantee your rational numbers are always in lowest terms?

   Hint: there are lots of ways to write this method. A simple way is to find the greatest common divisor (GCD) of the numerator and denominator (the largest integer that is a factor of both). As long as the GCD is greater than 1, you know the rational is not in lowest terms and you can divide both the numerator and denominator by this factor.

5. Add a method to your class that lets you multiply two rational numbers together. This function should take one rational number argument and **return** the product of the class's rational number with the argument:

   ```
   rational rational::multiply(const rational & other)
   ```

   Example of how this might be used:

   ```
   rational a(1, 2);
   rational b(3, 4);
   rational c = a.multiply(b); // a and b are unchanged, c is 3/8
   ```

   Hint: This is easier if you let your `reduce()` function do some of the work for you.

6. Add a method to your class that lets you add two rational numbers together. This function should take one rational number argument and **return** the sum of the class's rational number with the argument:

   ```
   rational d(2, 3);
   rational e(3, 4);
   rational f = d.add(f); // d and e are unchanged, f is 17/12
   ```

7. Add more capabilities to your class; e.g., conversion to a `double`, taking the inverse of a rational, allowing for negative rational numbers, subtraction, division, preventing division by zero, testing if one rational is less than another.