MAT 2170: Rational Class

and program to calculate π

The Rational Class

In mathematics, the rational numbers are defined as the set $Q = \{\frac{a}{b} \mid a, b \in Integers \wedge b \neq 0\}$. To develop a Java class to implement rational numbers, including the operations on them, we must first determine what invariants (rules), data members and member methods are necessary.

Invariants.

1. The denominator for the rational number is always positive. If the denominator is negative, the constructor transfers the sign to the numerator by negating that value.
2. The fraction represented by the numerator and denominator is always reduced to lowest terms by dividing both parts of the fraction by the greatest common divisor. This design decision ensures that rational numbers are always displayed in their simplest form.
3. `Rational` class should be immutable — once a `Rational` object is constructed, the client should not be allowed to modify it.

Data Members. A `Rational` class object will need to store a numerator and a denominator, both of type `int`. We shall need to guarantee they are always "in lowest (or reduced) terms" — in other words, that the numerator and denominator have no factors (other than 1 or −1) in common. Access to these data members should be restricted to the class itself — the client should not be able to manipulate the numerator or denominator directly.

Member Methods. When designing a class, it is necessary to consider what *constructor*s, *inspector*s, *mutator*s, and *facilitator*s are required or might be useful.

1. **Constructor**s:
   (a) `Rational()` — no parameters (aka *default* constructor), set to $\frac{0}{1}$
   (b) `Rational(n)` — a single parameter, set to $\frac{n}{1}$
   (c) `Rational(n, d)` — two parameters, set to $\frac{n}{d}$
   (d) copy constructor?
2. **inspector**s:
   (a) `getNumerator()` — returns the value of the numerator
   (b) `getDenominator()` — returns the value of the denominator
3. **mutator**s: — are not allowed for this class.
4. **facilitator**s will need to implement the usual arithmetic operations — add, subtract, multiply, and divide — the results of which are shown in the table below:

Rules for rational arithmetic	
Addition	Multiplication
$\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}$	$\frac{a}{b} * \frac{c}{d} = \frac{ac}{bd}$
Subtraction	Division
$\frac{a}{b} - \frac{c}{d} = \frac{ad-bc}{bd}$	$\frac{a}{b} \div \frac{c}{d} = \frac{ad}{bc}$

Also, the class should include the `toString()` method, which returns a string version of the fraction suitable for output. Another facilitator method should calculate the gcd(n, m) of two integers, for use in reducing the fraction.

Calculating π

There are many interesting ways to calculate π. For example[1],

$$\frac{\pi}{2} = 1 + \frac{1}{3} + \frac{1}{3}\cdot\frac{2}{5} + \frac{1}{3}\cdot\frac{2}{5}\cdot\frac{3}{7} + \cdots$$

or

$$\pi = 2\left(1 + \frac{1}{3} + \frac{1}{3}\cdot\frac{2}{5} + \frac{1}{3}\cdot\frac{2}{5}\cdot\frac{3}{7} + \cdots\right)$$

We would like a method, named `piApproximation`, which finds a *rational* approximation to π by using n terms of the series given above. For example, if $n = 2$, the result is $2(1 + \frac{1}{3}) = \frac{8}{3}$. This method has one input parameter — the desired number of terms — and returns one value, as a rational number. We wish to use a modified version of the `Rational` class in our solution, as follows:

- Add a new `public` method, `toDouble`, to the `Rational` class. This method will yield a floating point representation of a `Rational` object. For example, if a client program has declared `q` to be a `Rational` object, then the expression `q.toDouble()` is its floating point value.

Lastly, we need a client program which asks the user for n (the desired number of terms), and outputs the sum of the first n terms of the series given above in two ways: as an exact rational value, and then again as a floating-point value. If you implement this program, you will need to keep n relatively small when you run it — even with the use of `long`-type numerators and denominators, you will get quite a few digits fairly quickly[2].

[1] You can find this formula and many others at: `http://mathworld.wolfram.com/PiFormulas.html`. Calculating π is a mathematical problem of both historical and current interest. The references on the Wolfram site list dozens of articles about π, many written within the last twenty years.

[2] Exercise 10 (page 220) gives a hint how to solve this problem, using Java's `BigInteger` class. That is a more ambitious problem and is not required for this lab assignment.