class MyRational
  def initialize(num, den = 1) # second argument has a default
    if den == 0
      raise "MyRational received an inappropriate argument"
    elsif den < 0
      @num = -num # fields created when you assign to them
      @den = -den
    else
      @num = num # semicolons optional to separate expressions on different line
      @den = den
    end
    reduce # i.e., self.reduce() but private so must write reduce or reduce()
  end

  def to_s
    ans = @num.to_s
    if @den != 1
      ans += "/"
      ans += @den.to_s
    end
    ans
  end

  def to_s2
    dens = "" + @den.to_s
    if @den != 1
      @num.to_s + dens
    end
  end

  def to_s3
    "#{@num}#{if @den==1 then "" else "/" + @den.to_s end}"
  end

  def add! r
    # mutate self in-place
    a = r.num # only works b/c of protected methods below
    b = r.den # only works b/c of protected methods below
    c = @num
    d = @den
    @num = (a * d) + (b * c)
    @den = b * d
    reduce
    self # convenient for stringing calls
  end

  def gcd(x, y) # recursive method calls work as expected
    if x == y
      x
    elsif x < y
      gcd(x, y-x)
    else
      gcd(y, x)
    end
  end

  private
    def reduce
      if @num == 0
        @den = 1
      else
        d = gcd(@num.abs, @den) # notice method call on number
        @num = @num / d
        @den = @den / d
      end
    end
  end
end

def use_rationals
  r1 = MyRational.new(3, 4)
  r2 = r1 + r1 + MyRational.new(-5, 2)
  puts r2.to_s
  (r2.add! r1).add! (MyRational.new(1, -4))
  puts r2.to_s
  puts r2.to_s2
  puts r2.to_s3
  puts r2 # actually puts calls to_s on its argument
end

protected
  # there is very common sugar for this (attr_reader)
  # the better way:
  # attr_reader :num, :den
  # protected :num, :den
  # we do not want these methods public, but we cannot make them private
  # because of the add! method above
  def num @num end
  def den @den end