```
// Your job is to fill in all the missing function bodies. Be sure to add
your
// name to this file, and rename it!

/**
 * Rational is an object representing a "rational" number, or the ratio of 2
 * integers. Objects like this can be useful when you need to work with exact
 * quantities, without the rounding errors inherent in <code>double</code>s.
 *
 * @author      Adam Smith
 * @version     1.0
 */

class Rational {
    private int num, den;

    /**
     * Create a new <code>Rational</code>, from its numerator and
denominator.
     * @since 1.0
     */

    public Rational(int numerator, int denominator) {
    }

    /**
     * Add a second <code>Rational</code> to <code>this</code>.
     * @param other the other <code>Rational</code> to add
     * @return the resulting <code>Rational</code>
     * @since 1.0
     */

    public Rational add(Rational other) {
        return null;
    }

    /**
     * Subtract a second <code>Rational</code> from <code>this</code>.
     * @param other the other <code>Rational</code> to subtract
     * @return the resulting <code>Rational</code>
     * @since 1.0
     */

    public Rational subtract(Rational other) {
        return null;
    }

    /**
     * Multiply a second <code>Rational</code> by <code>this</code>.
     * @param other the other <code>Rational</code> to multiply
     * @return the resulting <code>Rational</code>
     * @since 1.0
     */

    public Rational multiply(Rational other) {
        return null;
    }
```

```
    /**
     * Divide <code>this</code> by a second <code>Rational</code>.
     * @param other the other <code>Rational</code> by which to divide
     * @return the resulting <code>Rational</code>
     * @since 1.0
     */

    public Rational divide(Rational other) {
        return null;
    }

    /**
     * Calculate the <code>double</code> whose value best approximates
     * <code>this</code>'s value.
     * @return the approximate value
     * @since 1.0
     */

    public double toDouble() {
        return Double.NaN;
    }

    /**
     * Build a <code>String</code> representing this <code>Rational</code>,
as
     * <code>(numerator/denominator)</code>.
     * @return the String <code>(numerator/denominator)</code>
     * @since 1.0
     */

    @Override
    public String toString() {
        return null;
    }

    /**
     * Return <code>true</code> if <code>this</code> <code>Rational</code> is
     * equivalent to another one; otherwise false.
     * @param otherObject the other object (probably a <code>Rational</code>)
to
     * compare with
     * @return the whether they are equivalent
     * @since 1.0
     */

    @Override
    public boolean equals(Object otherObject) {
        return false;
    }

    /**
     * Accessor for the numerator.
     * @return the numerator
     * @since 1.0
     */
```

```
    public int getNumerator() {
        return -1;
    }

    /**
     * Accessor for the denominator.
     * @return the denominator
     * @since 1.0
     */

    public int getDenominator() {
        return -1;
    }

    /**
     * Mutator for the numerator.
     * @param newNumerator the new numerator
     * @since 1.0
     */

    public void setNumerator(int newNumerator) {
    }

    /**
     * Mutator for the denominator.
     * @param newDenominator the new denominator
     * @since 1.0
     */

    public void setDenominator(int newDenominator) {
    }

    // helper function to calculate the Greatest Common Divisor
    private static int calcGCD(int int1, int int2) {
        return -1;
    }

    // helper method to reduce this Rational
    // call this in the constructor and the setters to reduce
    private void reduce() {
        // will call calcGCD() somewhere in here
    }
}
```