CSCI 3310 Assignment 2

100 points

Due date: Tuesday, Jan. 31 (before class)

For this assignment, you will be creating a class called **RationalNum** that will be used to store and manipulate rational numbers. A rational number is a number that is composed of two integers, where division of the integers is assumed. The division is not actually carried out; it is only indicated as in the following numbers:

1/2 2/3 15/32 65/4 16/5

You should represent rational numbers by using two integer *(int)* values, a numerator and a denominator. These two integer values should be stored as member variables in the class and should not be "publicly accessible".

You will need to write the following methods for the RationalNum class:

Constructors

All constructors should be defined as "public".

- A default constructor should be included. Remember that a default constructor is one that does not use any parameters. A rational number object created with the default constructor should be initialized with a zero (0) in the numerator, and a (1) in the denominator.

- A constructor should be included that receives a single integer as a parameter. Every integer number is also a rational number. For example, the integer number 2 is also the rational number 2/1. The integer number 17 is also the rational number 17/1. As a result, this constructor should place the given integer in the numerator, and the number one (1) in the denominator.

- A constructor should be included that receives two integer values as parameters. The first represents the numerator that should be stored in the RationalNum object and the second represents the denominator for the RationalNum object.

  If the constructor receives the value zero (0) to be put into the denominator, your constructor should print an error message that says: "Zero cannot be in the denominator".

  Negative rational numbers should always be indicated by placing a negative sign with the numerator. The denominator should always remain positive. As a result, this constructor should watch out for the possibility that a negative number might be passed in as the "denominator parameter". In other words, if we were asked to create the rational number (2 / -3), instead we should create the rational number (-2 / 3). The constructor might need to make adjustments to the data before storing the parameters into the member variables. Here is the general idea:

If the denominator parameter is positive, then the data is fine and therefore you can simply store the numbers given.

If the denominator parameter is negative, then change the sign of both parameters before storing them into the object. The result of this operation will produce either a negative or a positive rational number object. For example, the rational number (2 / -3) should be changed into the number (-2 / 3). The rational number (-2 / -3) should be changed into the number (2 / 3).

Overridden Methods

The following methods are inherited from the Object class, and you will need to overwrite the inherited versions with our own.

- **toString** - This method takes no parameters, and returns a String object. The purpose of this method is to return a string equivalent of the Rational Number object. The exact formatting returned should be the numerator, a slash, then the denominator. Do not add extra spaces or line breaks. So, if a particular rational number object was holding 5 as the numerator and 7 as the denominator, the toString method for this object should return the string "5/7"

- **equals** - This method receives an Object as a parameter, and returns a boolean value. If the Object that is passed as a parameter is not a RationalNum object, you should return false. Otherwise, perform a comparison using the current object's numerator and denominator along with the ones passed in through the parameter. Return true if the two rational numbers are equal, and return false otherwise. Note, do not simply compare the two numerators and the two denominators. Your comparison should be done in such a way as to show that 3/4 is equal to 6/8.

Public Methods

- **getDenominator** - This is a "getter" method used to retrieve the denominator. This method receives no parameters and returns an integer.

- **getNumerator**- This is a "getter" method used to retrieve the numerator. This method receives no parameters and returns an integer.

- **add** - This method should receive a single rational number as a parameter. It should add the parameter to the current *(this)* object and return the answer as a new rational number object. Neither the current object, nor the parameter should be changed by this method. For example, suppose x is the rational number "3/4", and y is the rational number "5/6",   and z is a rational number. The command `z = x.add(y)` should return the rational number "38/24" and store it in z.

x should not be changed by this method, and y should not be changed by this method. Refer to the formulas listed below to find out how to add two rational numbers.

- **sub** - This method should receive a single rational number as a parameter. It should subtract the parameter from the current *(this)* object and return the answer as a new rational number object. Neither the current object, nor the parameter should be changed by this method.
  For example, suppose x is the rational number "3/4", and y is the rational number "5/6", and z is a rational number. The command `z = y.sub(x)` should return the rational number "2/24" and store it in z. x should not be changed by this method, and y should not be changed by this method

- **mul** - This method should receive a single rational number as a parameter. It should multiply the current *(this)* object by the parameter and return the answer as a new rational number object. Neither the current object, nor the parameter should be changed by this method.

- **div** - This method should receive a single rational number as a parameter. It should divide the current *(this)* object by the parameter and return the answer as a new rational number object. Neither the current object, nor the parameter should be changed by this method.

- **neg** - This method will take no parameters. It should negate the current *(this)* object and return the answer as a new rational number object. The current object should not be changed by this method.
  For example, suppose x is the rational number "3/4", and z is a rational number. The command `z = x.neg()` should return the rational number "-3/4" and store it in z.

Formulas

Use the following formulas for the computations performed by the methods. In each of these formulas "A" is a numerator, and "B" is a denominator for one of the rational numbers; "C" is the numerator and "D" is the denominator for the second rational number. Also remember that since these are rational numbers, the division is not actually performed. Therefore, in the formulas, wherever you see a "division symbol", you don't actually perform the division; the "division symbol" is used in rational numbers to separate the numerator from the denominator. The only exception is on the left-side of the equals sign in the division formula where there is a division symbol used to indicate that we are trying to divide two rational numbers.

- Addition: `(A/B) + (C/D) = (A*D + B*C) / (B*D)`
- Subtraction: `(A/B) - (C/D) = (A*D - B*C) / (B*D)`
- Multiplication: `(A/B) * (C/D) = (A*C) / (B*D)`
- Division: `(A/B) / (C/D) = (A*D) / (B*C)`
- Negation: `-(A/B) = ((-A)/B)`
- Equality: `(A/B) == (C/D)` means `(A*D) == (C*B)`

Technical Notes

- At the top of EVERY Java file you create, you must include a comment which states your name.
- YOU MUST PROVIDE COMMENTS for EVERY method that is included in your class. It does not matter how simple the method is, you MUST still comment it. Follow the standard "javadoc" commenting style for formatting your comments.
- For this assignment, you need to write the rational number class and a demo class called **RationalNumDemo** to test all the methods in your RationalNum class.
- You do not need to finish writing the entire class before you can start testing it.
- Programs that do not compile will receive an automatic grade of "F".

Submission instructions:

To submit your programs, you need to submit your programs electronically on Blackboard. The detailed steps are as follows: go to the course Assignment page on Blackboard, click Assignment 2, and then scroll down and click "Browse my computer" to upload your program files. **Please also bring a hard copy of your programs to the class to submit.**

For the ease of grading your assignment, **please use a named package for each of your assignment.** For example, for assignment 2, please create a new package and **name your package as assg2_yourlastname** (with the first letter of your last name in uppercase and the rest in lower case), such as assg2_Smith. You also need to include a statement such as "package assg2_Smith;" at the beginning of each of your .java file. **Please follow this naming convention exactly for all future assignments. You will be deducted points for not doing so.**

When you submit your files to Blackboard, please submit your package folder (with source code only, i.e., .java files) as one zip file.