CSCI165 Computer Science II
Midterm Exam Project
Online Spring 2020

Define a class called Fraction. This class will be used to represent a ratio of two integers.

Properties:

- 2 private integers that will represent the numerator and denominator
- **Condition:** Our fractions will be positive only. No negative numerators or denominators allowed. If a negative value is passed in take it's absolute value and use that. **Do not duplicate this code. It should exist once and be called when needed.**
- **Condition:** Denominator can not be 0. If a denominator of zero is passed in it must be set to 1. **Do not duplicate this code. It should exist once and be called when needed.**

Constructors:

- No argument constructor that sets the numerator and denominator to 1
- An overloaded constructor that accepts a numerator and denominator
- The conditions stated above must be enforced at all times.

Methods: Method signatures **MUST MATCH THE FOLLOWING SPECIFICATIONS EXACTLY** or points will be taken off. The conditions stated above must be enforced at all times.

- Mutator methods (getters and setters) that allow the user to set the numerator and the denominator.
    - **setDenominator(int)**
    - **getDenominator()**
    - **setNumerator(int)**
    - **getNumerator()**
- A instance method called **toDecimal()** that returns the ratio as a double.
- An instance method called **reduce()** that reduces **this** fraction to lowest terms (e.g., if the fraction is 20/60, the method should reduce to 1/3). This will require finding the greatest common divisor for the numerator and denominator, then dividing both by that number.
- A **private** instance method called **gcd()** that will find and return the greatest common divisor between the numerator and denominator instance variables. Call this from reduce.
- A method **add(Fraction f)** that will accept a Fraction object as an argument and return a new fraction object that is the result of adding the argument to "this" Fraction
- A method called **subtract(Fraction f)** that will accept a Fraction object as an argument and return a new fraction object that is the result of subtracting the argument from "this" Fraction

- A method called **multiply(Fraction f)** that will accept a Fraction object as an argument and return a new fraction object that is the result of multiplying the argument by "this" Fraction
- A **toString()** method that will return a String representation of the Fraction ie. “1/2”. If the fraction is an improper fraction the toString must return it as a mixed fraction ie. “2 3/4”
- An **equals(Fraction f)** method that will accept a Fraction object as an argument and return true if the argument is equivalent to "this" Fraction or false if they are not equivalent. This is not just a simple test of numerator and denominator . . . you need to test the ratio of the two **(without explicit reduction to “this” fraction. Temporary reduction is fine)**
- A **compareTo(Fraction f)** method that will accept a Fraction object as an argument and return
    - 0 If the fractions are equal
    - – 1 if this fraction is less than f
    - +1 if this fraction is larger than f

Copy Constructor

There are many situations where you want to clone an instance. A cloned instance has the following properties

- original == clone returns false // identity: they aren’t the same object
- original.equals(clone) returns true // state: they are logically equal

In the same sense as the equals method, a direct assignment such as

original = clone;

is a **shallow copy, and DOES NOT create a new instance.** It simply copies the reference into a separate variable. We want to create a **new instance** that is an exact replica of an existing object’s state. This is called a **deep copy**. An easy way to achieve this is to define a **copy constructor.** A copy constructor accepts an argument of the class type, performs a deep copy of the object’s state and return a new instance. The signature for a copy constructor is

- **public ClassName(ClassName arg){**

    // field by field copy from arg to “this”

    }

Invocation of a copy constructor is as follows

- **ClassName clone = new ClassName(original);**

Define a copy constructor for the Fraction class such that you can clone a Fraction instance in the following way

- **Fraction original = new Fraction(1, 2);**
- **Fraction clone = new Faction(original);**

Application:

Create a Driver class that will demonstrate that your methods function properly. Accomplish this by creating multiple Fraction instances and calling each of the methods. Display appropriate messages that **CLEARLY** communicate what is happening.

You need to demonstrate each of the methods described above.

Reminder: To prove equivalence, and that a copy was successfully created show me that

- f1 == f2 returns false
- f1.equals(f2) returns true

Array of Fractions:

- Define the following two arrays in the Driver
  - int[] numerators = {1, 2, 3, 4, 11, 6, 7, 8, 13};
  - int[] denominators= {2, 5, 4, 6, 3, 8, 21, 13, 5};
- Create an array of 9 Fraction objects where **numerators[index] and denominators[index]** form the fraction. For example . . . the first 3 fractions will be: ½, 2/5 and ¾
- Using a loop iterate through the array, call reduce and print the toString.
- Write code to determine the smallest and largest fractions in the array. Print these to the terminal with a descriptive message.

Unit Tests:

You have been provided with a JUnit test file. Run these tests against your class. All tests must pass in order for you to receive full credit. You may not modify any of the tests unless you can prove to me that they are incorrect. Take a screenshot of the JUnit tests passing and understand that I will be running my test file against your code.

Submission:

Push your **Unit test screen shot**, **Fraction.java** and **Driver.java** source files only. **Do not submit compiled class files**

Be sure to carefully review your code against the requirements. Also understand that this is an exam and you are not allowed to receive help. I expect you to be able to describe, in detail, what your code is doing.