CSD 3464 – ASSIGNMENT 02 (Question 02)

Overview:

The following question is an extension to *Absolute Java* (6th Ed.)'s Chapter 04 Programming Project Q2 (pg. 253). Please follow the instructions included in **this** document and implement the following Java files:

⇒ FractionDemo.java *(Contains a `main()` method)*
⇒ Fraction.java

The above classes/files should both be inside a package called *q2*.

Instructions:

Create a `Fraction` class with `private` fields that hold a positive `int numerator` and a positive `int denominator`. In addition, create accessors and mutators for each field; with respect to the mutators, ensure `numerator` is greater than or equal to 0 and the `denominator` is greater than 0 (illegal values should be set to 1).

The `Fraction` class should have two constructors: the first should be a no-parameter constructor which sets the `numerator` to 0 and the `denominator` to 1 while the second should take two parameters, one for the `numerator` and one for the `denominator`. Be sure to `reduce()` the fraction to its simplest form when the fraction is created or altered.

In terms of methods, the `Fraction` class should have a `private` method `reduce()` which reduces the fraction to its simplest form (e.g., 3/6 should be reduced to 1/2). The reduction can be performed by finding the largest value that can divide evenly into both the `numerator` and the `denominator` (suggestion: use a `for` loop and starting with the smaller of the `numerator` or `denominator`).

Another method that is to be included is an overridden `public` method for `toString()` which returns a string that represents the fraction (i.e., "3/4"). Please note that `toString()` does not print out the fraction but simply returns a string that can be printed when a `Fraction` object is referenced in a `WriteLine` statement. For example: if `f1` is an object of type `Fraction` containing a `numerator` of 3 and a `denominator` of 4, then the statement `System.out.println("The fraction is " + f1)` would output:
`The fraction is 3/4.`

Also include a multiplication method: i.e., `public multiply()`. This method multiples two `Fraction` objects and returns a `Fraction` object. Recall to multiply fractions, multiply the `numerator`s and multiply the `denominator`s. Be sure to also `reduce()` the result.

In addition to multiplication, you are also required to include an addition method, i.e., `public add()`. This method adds two `Fraction` objects and returns a `Fraction` object. Recall

to add two fractions, you first need a common `denominator`. While there are several approaches to achieve this, the simplest way is to multiply the `numerator`s of the two fractions by their opposite `denominator`s. Then add the two `numerator`s to create the `numerator` for the new fraction with its `denominator` being the product of the operand `denominator`s. Be sure to `reduce()` the result.

For example: 3/8 + 1/4 = (3x4)/(8x4) + (1x8)/(4x8) = 12/32 + 8/32 = 20/32 ... which when reduced is 5/8.

Your class must also include the method `public equals()` which compares two `Fraction` objects and returns the boolean value `true` if and only if both the `numerator` and `Denominator` of the two `Fraction` objects are the same.

The final methods to include are `public greaterThanOrEqualTo()` and `public lessThanOrEqualTo()` methods that compare two `Fraction` objects and return a boolean (`true/false`) value. Since fractions may have different `denominator`s, the simplest way to compare two fractions is to compute the floating-point value for each fraction and then compare. To compute the floating-point value, divide the `numerator` by the `denominator`. For example, if the fraction was 3/4, then `double val = 3 / (double) 4;` would assign `0.75` to `val` (notice the explicit casting to avoid the perils of integer division: `val = 3 / 4;` would assign `0` to `val`).

You are also required to write a `static` class called `FractionDemo` in another file that contains a `main()`. The purpose of this class is to test the functionality of your `Fraction` class. It is up to you to decided what to include in the `main()` to test the class; however, all `public` facing methods and constructors need to be tested for correctness. Additionally, you should test to ensure invalid entries (`num` and `den`) passed to the constructors and mutator methods are correctly handled.

The basic structure of your `Fraction` class is required to look like the following:

```
public class Fraction
{

     private int numerator;
     private int denominator;

     // No Parameter Constructor
     public Fraction()
     {
        // insert code here
     }

     // Two Parameter Constructor
     public Fraction(int num, int den)
     {
```

```
    // insert code here
}

// Mutator - Numerator
public void setNumerator(int num)
{
    // insert code here
}

// Accessor - Numerator
public int getNumerator()
{
    // insert code here
}


// Accessor - Denominator
public void setDenominator()
{
   // insert code here
}


// Mutator - Denominator
public int getDenominator()
{
   // insert code here
}

//Reduce method
private void reduce()
{
   // insert code here
}

// toString method
@Override
public String toString()
{
   // insert code here
}

// Multiply method
public Fraction multiply (Fraction fract2)
{
   // insert code here
}

// Add operator
public Fraction add (Fraction fract2)
{
   // insert code here
```

```
    }

    // equals method
    @Override
    public boolean equals(Fraction fract2)
    {
       // insert code here
    }

    // Greater Than or Equal
    public bool greaterThanOrEqualTo(Fraction fract2) {

       // insert code here
    }

    // Less Than or Equal
    public bool lessThanOrEqualTo (Fraction fract2) {

       // insert code here
    }

}
```

UML Diagram:

The below UML diagram is provided for your convivence. Ensure all shown fields, constructors, and methods are included in your Fraction class prior to your final submission.