# Trent University

COIS1020H
Winter 2019
**Assignment 5**
Due: April 5, 2019

Create a `Fraction` class with private fields that hold a positive `int numerator` and a positive `int denominator`. In addition, create Properties for each field with the `set` mutator such that the `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

```
      Console.WriteLine("The fraction is {0}, f1)
```

would output: `The fraction is 3/4`. Note the use of `f1` as oppose to `f1.ToString()` as the latter is redundant.

Also include an overloaded multiplication operator: i.e., `public operator*()`. This method multiples two `Fraction` objects and returns a `Fraction` object. Recall to multiply fractions, multiply the `numerators` and multiply the `denominators`. Be sure to also `Reduce()` the result.

In addition to multiplication, you are also required to overload the addition operator, i.e., `public operator+()`. 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 `numerators` of the two fractions by their opposite `denominators`. Then add the two `numerators` to create the `numerator` for the new fraction with its `denominator` being the product of the operand `denominators`. 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.

The final methods to include are `public operator>=()` and `public operator<=()` methods that compare two `Fraction` objects and return a boolean (`true/false`) value. Since fractions may have different `denominators`, 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. `Main()` is to include an array called `testFractions` that contains five (5) `Fraction` objects. `testFraction[0]` is to be created using the two-parameter constructors (use a `denominator` of 1 and a `numerator` of 2), while `testFraction[1]` is to be created using the no-parameter constructor. You are the to prompt the user to enter values for `testFraction[1]` and assign it values using the properties. Before you move on, be sure to have your program print out the values of `testFraction[0]` and `testFraction[1]`.

Now you are prepared to demonstrate that your overloaded operators function properly. `testFraction[2]` is to hold the result from adding `testFraction[0]` to `testFraction[1]` while testFraction[2] is to store the result from multiplying `testFraction[0]` to `testFraction[1]`. Finally, `testFraction[4]` is to hold the result of `testFraction[2]` added to `testFraction[3]` and then multiplied by `testFraction[0]`. Your program should now print out the values of `testFraction[2]`, `testFraction[3]` and `testFraction[4]`.

Finally, `Main()` should also compare the fraction objects using to show that both relational operators (>= and <=) work properly.

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
        }
        // Numerator Property
        public int Numerator
        {
                // insert code here
        }
        // Denominator Property
        public int Denominator
        {
                // insert code here
        }
        //Reduce method
        private void Reduce()
        {
                // insert code here
        }
        // ToString method
        public override string ToString()
        {
                // insert code here
        }
        // Multiply method
        public static Fraction operator*(Fraction fract1, Fraction fract2)
        {
                // insert code here
        }
        // Add operator
        public static Fraction operator+(Fraction fract1, Fraction fract2)
        {
                // insert code here
        }
        // Greater Than or Equal operator
        public static bool operator>=(Fraction fract1, Fraction fract2)
        {
                // insert code here
        }
        // Less Than or Equal operator
        public static bool operator<=(Fraction fract1, Fraction fract2)
        {
                // insert code here
        }
    }
```

For this assignment, you are to submit two properly documented source code files (one .cs file for EACH class). Both source code files are to include comments at the top containing your name, student number, a description of the class, and comments within the body of your class (inline comments). You are also required to summit a PDF of your testing documentation using the template that is provided on BlackBoard. Failure to use the testing template or submit as a PDF could result in a 0 for the testing component of the assignment. These 3 files are to be attached to the Assignment 5 DropBox by the due date.