Exercise 2: Fraction

1. Requirement:

a) Business Requirements:

The "Fraction" class is designed to handle fractional numbers represented by a numerator and a denominator. The goal of this project is to provide a flexible and reliable data structure for performing arithmetic operations with fractions, such as addition, subtraction, multiplication, and division. The class should also allow the simplification of fractions and comparison between different fractions to determine equality. The "Fraction" class will be used in various applications where precise fractional calculations are required, such as in financial calculations, engineering, and scientific computations.

b) Technical Programming Requirements:

Design the **Fraction** Class with the private attributes to store the numerator and denominator and methods as follow:

- Provide a **default constructor** without any arguments to create a fraction with a value of **0/1**
- Provide a **constructor** to initialize the Fraction Class with a numerator and a denominator.
- Implement **getter** and **setter** methods to access and update the numerator and denominator. (Handle negative fractions properly in setters)
- Provide a **toString** method to represent the fraction as a string (Negative fractions should be represented correctly with a minus sign)
- Implement methods to perform basic operations (**addition**, **subtraction**, **multiplication**, **division**) and **simplify** fractions.
- Implement a method to **compare** fractions and determine their equality.
- Handle special cases like negative fractions and invalid fractions (denominator equal to 0). (Handle negative fractions in class initialization, setters, and toString representation)
- Create a test class named **TestFraction** to test various operations and scenarios involving fractions. The test class should include test cases to cover positive and negative fractions, basic operations, simplification, comparisons, and handling of special cases. Ensure the test class demonstrates the correct functioning of the Fraction Class and provides clear output for validation.

You should implement the Fraction class base on the class diagram as below:

Fraction
- numerator: int - denominator: int

+ Fraction(numerator: int, denominator: int) + getNumerator(): int + setNumerator(numerator: int): void + getDenominator(): int + setDenominator(denominator: int): void + toString(): String + simplify(): void + add(fraction: Fraction): Fraction + subtract(fraction: Fraction): Fraction + multiply(fraction: Fraction): Fraction + divide(fraction: Fraction): Fraction + equalTo(fraction: Fraction): int

c) Example:

With positive fraction initialization and basic operations:

- Suppose we have two fractions: a = 2/4 and b = 3/6.
- Adding two fractions: a + b = (2/4) + (3/6) = (2*6 + 4*3)/24 = 24/24 = 1/1
- Subtracting two fractions: a - b = (2/4) - (3/6) = (2*6 - 4*3)/24 = 0/24 = 0/1
- Multiplying two fractions: a * b = (2/4) * (3/6) = (2*3)/24 = 6/24 = 1/4
- Dividing two fractions: a / b = (2/4) / (3/6) = (2*6)/(4*3) = 12/12 = 1 /1
- Comparing two fractions: a = b => True

With positive fraction initialization and basic operations:

- Suppose we have two fractions: a = -2/4 and b = 3/(-6) => **b = -3/6**
- Adding two fractions: a + b = (-2/4) + (-3/6) = [(-2)*6 + (-3)*4] / 24 = -24/24 = -1/1
- Subtracting two fractions: a - b = (-2/4) - (-3/6) = [(-2)*6 - (-3)*4] / 24 = 0/24 = 0/1
- Multiplying two fractions: a * b = (-2/4) * (-3/6) = [(-2)*(-3)] / 24 = /1/4
- Dividing two fractions: a / b = (-2/4) / (-3/6) = [(-2)*6] / [(-3)*4] = 1 /1