Rational Number Class Part 1

C# has several numeric types, including natural, real, and irrational numbers. One numeric type that's missing is a Rational number. A rational number, as the name suggests is a ratio between 2 natural numbers and is usually represented as a fraction in the form 1/2, 5/4, -79/13, etc.

Create a C# struct with a constructor that takes two integer parameters, either or both of which may be positive or negative. Include two read-only properties, Numerator and Denominator, that return the numerator and denominator of the fraction respectively of type int. Also, override the ToString() method to give a string representation of the rational number as described in the preceding paragraph.

Examples:

```
var r1 = new Rational(1, 2);
r1.Numerator ➞ 1
r1.Denominator ➞ 2
r1.ToString() ➞ "1/2"

var r2 = new Rational(10, 8);
r2.Numerator ➞ 5
r2.Denominator ➞ 4
r2.ToString() ➞ "5/4"

var r3 = new Rational(2, -1);
r3.Numerator ➞ -2
r3.Denominator ➞ 1
r3.ToString() ➞ "-2"

var r4 = new Rational(-16, -64);
r4.Numerator ➞ 1
r4.Denominator ➞ 4
r4.ToString() ➞ "1/4"
```

Notes:

- The numerator may be zero, but if the denominator is zero an ArgumentException should be raised by the constructor function.
- The Numerator and Denominator values should be reduced to their lowest possible integer values to maintain the ratio (examples r2 and r4 above).
- If the resulting fraction is a whole number, the Denominator should return 1 but the ToString() method should only show the integer value (example r3 above).
- If one of the values of numerator and denominator is negative, the sign is shown on the Numerator and the Denominator is positive (example r3 above).
- If both the numerator and denominator are negative, the fraction is positive and both Numerator and Denominator are positive (example r4 above).
- If the numerator is zero, the denominator should be set to 1, regardless of the value passed to the constructor.

Rational Number Class Part 2

C# has several numeric types including natural, real and irrational numbers. One numeric type that's missing is a Rational number. A rational number, as the name suggests, is a ratio between two natural numbers and is usually represented as a fraction in the form 1/2, 5/4, -79/13, etc.

In part 1 of this challenge, you were asked to create a C# struct with a constructor that takes two integer parameters, either or both of which may be positive or negative. The struct should include two read-only properties, Numerator and Denominator, which return the numerator and denominator of the fraction respectively of type int. These properties should be reduced to their lowest common form. The ToString() method should be overridden to give a string representation of the rational number as described in the preceding paragraph.

Task: Your task now is to add arithmetical operations to your class so that rational numbers can be added, subtracted, multiplied, and divided. To do this you will need to understand how to overload the operators +, -, *, and /. You should also implement the unary '-' operator.

Since Math.Sign() does not know about the Rational class, implement a read-only property Sign that returns an int value of -1, 0 or '1' depending on the sign of the fraction.

Examples:

```
var r1 = new Rational(1, 2);

var r2 = new Rational(10, 8);

var r3 = new Rational(2, -1);


// The following operations are valid:

var r4 = r1 + r2; r4.ToString() → "7/4"

(r1 * r3).ToString() → "-1"

(r2 - new Rational(-1, 4)).ToString() → "3/2"

var r5 = (r1 + r2) / r3; r5.ToString() → "-7/8"

// Unary operator '-' changes the sign

(-r1).ToString() → "-1/2"

(-r3).ToString() → "2"

(r1 - -r2).ToString() → "7/4" (1/2 + 5/4)

r3.Sign → -1
```

Notes:

- You need to cope with negative, positive, or zero rational numbers.
- If an attempt is made to divide by a zero rational number (e.g. new Rational(0, 1)), a DivideByZeroException should be raised.

Rational Number Class Part 3

In parts 1 and 2 of this 3-part challenge, you were asked to develop a Rational number struct. A rational number, as the name suggests, is a ratio between two natural numbers and is usually represented as a fraction in the form 1/2, 5/4, -79/13, etc.

In part 1, the task was to create a C# struct with:

- A constructor which takes two integer parameters,
- two read-only properties Numerator and Denominator which return the numerator and denominator of the fraction respectively, reduced to their lowest common form.
- An override of the ToString() method giving a string representation of the rational number as described in the preceding paragraph.

In part 2, the code was extended to:

- Allow arithmetic operations +, -, *, and / between rational numbers (including unary - operator)
- A Sign property that returned the sign of the fraction.
- Before completing this part, make sure your class meets the conditions described in parts 1 and 2. The best way to do this is to complete those challenges first and use the code to start this challenge.

Task: Your task now is to add comparison operators to your class so that rational numbers can be compared with each other using the comparison operators ==, !=, <, <=, > and >=. A further requirement is to implement implicit and explicit conversion to and from the C# decimal value type.

Examples:

```
var r1 = new Rational(1, 2);
var r2 = new Rational(10, 8);
var r3 = new Rational(2, -1);
// Comparison operations
r1 == r2 ➞ false
r2 >= r1 ➞ true
r2 != r3 ➞ true
r1 == new Rational(17/34) ➞ true
// Type conversion
var r4 = (Rational)6.5572m; ➞ explicit conversion from decimal
var d = (decimal)r2;  ➞ explicit conversion to decimal
r4 == 6.5572m ➞ true (implicit conversion to decimal)
r4.ToString() ➞ "16393/2500"
```

Note: When overloading comparison operators == and !=, you'll also be expected to override the Equals() and GetHashCode() methods.

Credit: https://edabit.com/collection/Z74Cmxhbk5W5cqkrh