Lab17: Fraction Class

Part 1: Create a Fraction class with the following comments:

```
// Constants (Final Static Variables)
// Static/Class Variables
// Nonstatic/Instance Variables
// Constructors
// Getters/Setters
// Static Methods
// Nonstatic Methods
```

Under the appropriate comments, create the following:

```
private static int objectCount=0; // Number of Fractions created
private static int methodCount=0; // Number of times methods have been called.
private int numerator;
private int denominator;
```

Part 2: Create getters for all 4 variables in Part 1. Create setters for numerator and denominator.

Part 3: Create a toString() method:

```
// Returns a nice looking fraction like "3/4". Your fractions should look nice, but does not need to be simplified.
public String toString();
```

Part 4: Create the following 3 constructors, remember to count each object:

```
public Fraction()

public Fraction(int n, int d)

// This creates the Fraction as a copy of another Fraction.
public Fraction(Fraction f) //Ex, Fraction f = new Fraction(new Fraction(1,2));
```

Test Your Code!

In Main.main(), create the fractions 1/1, 1/2, 1/3, ... , 1/90 and print them out, one per line.

Part 5: Create the following methods, remember to count your methods:

```
// Returns in mixed number format. Ex, 5/3 => "1 2/3".
// Please consider all cases! Ex, "30 1/2", "4", "1/51"
public String toMixedNumber();
```

```
// The integer part of 25/3 is 8.
public int integerPart();
public static int integerPart(Fraction f);
```

```
// The remainder of 7/5 is 2.
public int remainder();
```

```
// Returns true if it's divide by 0 error.
public boolean isError();
```

```
// Absolute value.
public static int abs(int a);
```

```
// Return true if it's positive, false otherwise.
public boolean isPositive();
```

```
// Returns true if the fractions have the same value, false otherwise.
// Ex, f = 2/4, g = 3,6, f.equals(g) => true.
public boolean equals(Fraction f);
```

```
// Return the smaller/bigger of the 2 Fractions. Return copies, not pointers to
the original!
public static Fraction min(Fraction a, Fraction b);
public static Fraction max(Fraction a, Fraction b);
```

Test Your Code! Paste and run this code in Main.main().

```
Fraction f = new Fraction(26,3);
Fraction g = new Fraction(4,0);
Fraction h = new Fraction(3,-7);
System.out.println("8 2/3:\t" + f.toMixedNumber());
System.out.println("8:\t" + f.integerPart());
System.out.println("2:\t" + f.remainder());
System.out.println("true:\t" + g.isError());
System.out.println("false:\t" + h.isPositive());
System.out.println("26/3:\t" + Fraction.max(f,h).toString());
```

Part 6: Fraction methods that require a more complex algorithm:

```
// To determine if a fraction is simplified, look at values 2,3,4,...,
// and see if any of them divide both the numerator and denominator.
// Ex, 6/15 is not simplified because 3 divides both 6 and 15.
// If both the numerator and denominator are negative,
// the Fraction is not simplified.
public boolean isSimplified();
```

```
// To simplify a fraction, go through all numbers >=2, and if it divides
// both numerator and denominator, divide it out of the numerator and
// denominator. Consider what happens when you have 4/8!

// Also, account for positive and negative numerator/denominator.
// If the Fraction is positive, make both numerator and denominator be +.
// If the Fraction is negative, make the numerator - and denominator +.

// The simplify() function should simplify the Fraction you are calling
// it from.
public void simplify();
```

```
// The simplified() function should return a new simplified version of the
// Fraction, and not change the original Fraction.
public Fraction simplified();
```

Part 7: Arithmetic

```
// Returns the sum of this Fraction and a.
public Fraction sum(Fraction a);
```

```
// Returns the sum of all Fractions in an array of Fractions. Use sum().
public static Fraction sum(Fraction[] a);
```

```
// Returns the difference of this Fraction minus a.
public Fraction difference(Fraction a);
```

```
// Returns the product of this Fraction times a.
public Fraction product(Fraction a);
```

```
// Returns the quotient of this Fraction divided by a.
public Fraction quotient(Fraction a);
```

```
// Returns the square root of this Fraction, using your sqrt() method from
// the previous lab and rounding down.
// Paste your MathClass class into this project to use your sqrt() method.
// You may round down the numerator and denominator separately for simplicity.
public Fraction sqrt();
```

Part 8: Quadratic Formula.

The roots of a degree 2 polynomial of the form: $y = ax^2 + bx + c$
can be found using the quadratic formula $x = (-b +- sqrt(b^2 - 4ac))/(2a)$

```
// Returns the roots of a degree 2 polynomial where a,b,c are Fractions.
// Because there may be 2, 1, or 0 solutions, the return value should be
// a 2 element Fraction array, a 1 element Fraction array, or null.
public static Fraction[] quadraticFormula(Fraction a, Fraction b, Fraction c);
```

Test your code! In Main.main(), test your code on $y = 2x^2 + 3x + 1$, $y = 4x^2 - 9$, and $y = x^2 + 1$.

```
// Include your code and test code.
```

Part 9: "Partial" Partial Fraction Decomposition

What we are trying to code: Any fraction between 0-1, whose denominator is factorable, can be written as the sum of fractions whose denominators are the factors of the denominator of the original fraction, and the numerators are natural numbers (positive integers or zero).

Ex, 5/12 = A/2 + B/3 + C/4 + D/6 + E/12 = 0/2 + 1/3 + 0/4 + 0/6 + 1/12

Ex, 2/15 = A/3 + B/5 + C/15 = 0/3 + 0/5 + 2/15

Helper functions:

```
// Returns the factors of n, not including 1 and including n.
public static int[] factors(int n);

// Returns an array of Fractions whose sum is the Fraction you are in.
// You may either include or exclude zero Fractions.
// Ex, 5/12 can return {0/2, 0/3, 1/4, 1/6, 0/12} or {1/4, 1/6}, your choice.
public Fraction[] partialPartialFractionDecomposition();
```