Assignment 1

Fraction Class	70 pts.
AggregationSample Class	30 pts.

TOTAL: 100 pts.

General Requirements

- *All files you are turning in must be signed. Please add your name to the source code in comments. Unsigned work automatically loses **5 pts**.*
- *Add comments to the source code you are writing:*
  - *Describe the purpose of every variable*
  - *Explain the algorithm you are using for solution*
  - *Add proper comments for all methods. Include @param, @return, and @throws tags*
- *Uncommented work loses up to **25% of the grade.***
- *Archive the entire project using ZIP or RAR utility. Turn it in into the digital drop box.*

Fraction Class

Implement a class called Fraction. The class's purpose is to store a representation of a fraction. Fraction's numerator and denominator values stored in class fields as integers. If the fraction is negative, the negative sign is recorded as part of numerator. The fraction is always stored in its simplest form.
The class should have the following methods:

- **Constructor 1** that accepts two integer numbers, numerator and denominator.
  - Denominator must not be a 0. If a 0 passed as a denominator, an `IllegalArgumentException` must be thrown by the constructor.
  - Constructor must call simplify() method to simplify fraction after setting the fields.
- **Constructor 2** accepts three integer numbers to represent mixed fraction: first number is the whole part the other two are numerator and denominator. Internally the fraction is still stored as numerator and denominator only.
  - Denominator must not be a 0. If a 0 passed as a denominator, an `IllegalArgumentException` must be thrown by the constructor. Example:
  - `Fraction n = new Fraction(25, 3, 5); // mixed fraction 25 and 3/5`
  - Constructor must call simplify() method to simplify fraction after setting the fields.

- **Non-argument constructor**. The constructor must set numerator to 0 and denominator to 1.
- **Copy constructor** . Sample copy constructor call:

  -
    ```
    Fraction a  = new Fraction (1, 3); // regular
                                        // constructor call
    Fraction d = new Fraction (a); // copy
                                        // constructor call
    ```
- **Accessor and mutator** methods for numerator and denominator fields.
  - When mutator method used to change the denominator of the Fraction object, and a 0 is passes as an argument, the method must throw an `IllegalArgumentException`.
  - Mutators must call simplify() method to simplify fraction after change.
- **Addition method** `Fraction add(Fraction obj)`. When two Fraction objects are added together, the result is a Fraction object that is the sum of two fractions. Do not forget that in order to add two fractions together you first need to convert them to have common denominator. In order to simplify the coding process for you, the sample code that adds fractions is provided. See it in **GCD.java**. I would recommend creating a private method that finds and returns greatest common divider (GDC). Please see sample code that calls add() method below:
  ```
  Fraction a  = new Fraction (1, 3);
  Fraction b = new Fraction (1, 12);
  Fraction c = a.add(b); // c = a+b
  ```
  - simplify() method must be called to simplify fraction after changing the fields.

- **Subtraction method**. `Fraction subtract(Fraction obj)`. When two Fraction objects are subtracted from each other, the result is a Fraction object that is the difference of two fractions. Logic here is the same as in fraction addition. Please use greatest common divider.
  - simplify() method must be called to simplify fraction after changing the fields.

- Private method **simplify()** simplifies the fraction as much as possible. This can be done by finding greatest common divider for numerator and denominator. A sample of this operation can be found in **GCD.java**. Use **simplify()** method after you do any arithmetic operations on fraction. Fraction should always be stored in its simplest form.
  Example : 32/68 = 8/17
- **Multiplication.** `Fraction multiply(Fraction obj)` Multiplies two fractions then <u>simplifies</u> the result.

- simplify() method must be called to simplify fraction after changing the fields.

- **Division.** `Fraction divide(Fraction obj)` Divides two fractions then simplifies the result.
  - simplify() method must be called to simplify fraction after changing the fields.
- **`String toString()`** method that converts the fraction into a string. For example "*3/22* ".
  - If the fraction is more than 1 (has a whole part), like 25/24, in the output it must be converted into a mixed fraction: "*1 and 1/24"*
- **Less**. `boolean less(Fraction obj)`. Method returns true if the calling fraction object is smaller than the parameter object. Do not forget to convert both fractions so they have a common denominator before you compare them.

  -
    ```
    Fraction a  = new Fraction (1, 3);
    Fraction b = new Fraction (1, 12);
    boolean res = a.less(b); // res is holding false
    because 1/3 is NOT less than 1/12
    ```
- **More** `boolean more(Fraction obj)`. Similar to `less` method
- **Equals** `boolean equals(Fraction obj)`. Again, here you need to get the fractions to the common denominator in order to compare them.

Name your project `Fraction.java`

Test <u>all the methods</u> you wrote in main. Make sure to have the code that catches `IllegalArgumentException` and handles it in main(). Place the main() in separate file named FractionDemo.java.

AggregationSample Class

In order to gain practical knowledge of aggregation and to practice deep copying of the field objects we will implement a class called AggregationSample.
The class must have 3 fields:

- fractionField of type Fraction (class from the first problem)
- arrayField – array of integers - type int[]
- doubleField of type double

The class does not really have any functionality and its main purpose is to give you a hands-on review of the aggregation concept. Class has a very limited number of methods, all of them are standard methods that you would expect to have in any regular class.

1. Non-argument constructor that sets
   a. fractionField to fraction 0,

   b. arrayField to an array of size 3 filled with 0s, and
   c. doubleField to 0
2. Constructor that takes a fraction (as an object), an array, and a double as parameters and sets all 3 fields to the given values. Make sure to create DEEP COPIES of all non-primitive parameters when setting the new object fields.
3. Copy constructor. Make sure to create DEEP COPIES of all non-primitive fields when setting the new object fields.
4. Three accessor methods for all 3 fields. Make sure to create DEEP COPIES of all non-primitive fields and return a reference to a copy not to the original field.
5. Three mutator methods for all 3 fields. Make sure to create DEEP COPIES of all non-primitive parameters when changing the value of the fields.
6. toString() method that creates and returns a string containing all 3 fields in text format, like "*Fraction: 1 and 1/24; Array: [1,0,5]; Double: 3.45*"

Use tests provided in **AggregationDemo.java**. Please make sure all tests run and produce expected results.