Assignment #2

Due: Fri, Jan 29

Build a class called `IntegerSet`, which is based on Exercises 8.13 and 8.14 from the textbook, noting more specific instructions below. Also, add features into the Fraction class (the example discussed in lecture) as described below.

Filenames should be

- `IntegerSet.java`
- `Fraction.java`

Note that this filename starts with a capital letter. Please make sure your filenames match mine **exactly**.

Instructions

IntegerSet

Short summary of exercise (paraphrased from the book), along with extra instructions:

- Create class `IntegerSet`
- An `IntegerSet` object holds integers in the range 0-100
- Represented by an array of `boolean`s, such that array element `a[i]` is set to `true` if integer *i* is in the set, and false otherwise
- Create these constructors and methods for the class
- `IntegerSet()`
- `public IntegerSet union(IntegerSet iSet)`
- `public IntegerSet intersection(IntegerSet iSet)`
- `public IntegerSet insertElement(int data)`
- `public IntegerSet deleteElement(int data)`
- `public boolean isEqualTo(IntegerSet iSet)`
- `public String toString()`
- The constructor (no arguments) initializes the array to represet the "empty set" (i.e. no integers in the set)
- Method `union` creates and returns a new set that is the set-theoretic union of the two existing sets (the calling object and the parameter). An element is in the union if it's in either of the starting two sets
- Method `intersection` creates and returns a new set that is the set-theoretic intersection of the two existing sets (the calling object and the parameter). An element is in the intersection if it's in BOTH of the starting two sets

- Method `insertElement` adds the argument (an integer) to the set (the calling object), and also should return that set (so that calls can be cascaded)
- Method `deleteElement` removes the argument (an integer) from the set (the calling object), and also should return that set (so that calls can be cascaded)
- Method `isEqualTo` determines whether two sets are equal (i.e. they have all the same elements), returning a true or false indication
- Method `toString` returns a string containing the set elements as a list of numbers, in ascending order, separated by spaces. Include only elements present in the set. Use "---" to represent an empty set.

Fraction exercise

Start with this copy of the Fraction class, which was discussed in lecture class (download from the link here), and add the following features:

- Write a method called `Simplify`, which returns a simplified version of the calling object (no parameters are needed). The method should return a new Fraction (simplified), but not change the original one. The fraction in the form 0/N should have simplified form 0/1. Any other fraction has the usual mathematical definition of "simplified form". Keep within the rules already extablished in this Fraction class (e.g. denominator always positive, any negative values go in the numerator, etc).

- Write methods `add`, `subtract`, `multiply`, and `divide`. Each one should take in a Fraction as a parameter and perform the given computation between the calling object and the parameter object. (The calling object is always the first operand). The result of each operation should always be a fraction returned in simplified form. Example calls:
- `  f1.add(f2)              // means f1 + f2`
- `  f1.subtract(f2)         // means f1 - f2`

  In `divide`, if an attempt is made to divide by a fraction with the value 0, default the result to 0/1. (Such division is actually undefined. 0/1 is not the "true" result, but we need to return something from this method).

- Here are your method signatures:
- `public Fraction simplify()`
- `public Fraction add(Fraction f)`
- `public Fraction subtract(Fraction f)`
- `public Fraction multiply(Fraction f)`
- `public Fraction divide(Fraction f)`
- Make sure that your new methods enforce the same rules on the data that my original ones do -- the denominator must be non-negative (any negative sign goes in the numerator) and the denominator must never be zero (this would be undefined).

Testing

I've provided a file to help you get started with testing. This is not a comprehensive set of tests (so you'll want to try some of your own). However, this will get you started. And you **do** need to include the HW2Tester class (unchanged) in your jar file when you submit. See instructions at the bottom.

Here's the HW2Tester.java file. This contains some tests for both the IntegerSet and the Fraction classes.

Here's the sample run

```
myers@diablo:~>java HW2Tester

After set1.insertElement(10), set1 = 0 2 8 10
default IntegerSet is = ---
set1 = 0 2 4 6 8 10 12 95 100
set2 = 0 3 6 9 12
set1.union(set2) = 0 2 3 4 6 8 9 10 12 95 100
set1.intersection(set2) = 0 6 12
set1.deleteElement(2) = 0 4 6 8 10 12 95 100
set1.isEqualTo(set1) = true
set1.isEqualTo(set2) = false


Fraction tests:

4/6 simplified = 2/3
75/175 simplified = 3/7
-6/17 simplified = -6/17
f1 = 4/6
f2 = 75/175
f3 = -6/17
4/6 + 75/175 = 23/21
4/6 - 75/175 = 5/21
4/6 * 75/175 = 2/7
4/6 / 75/175 = 14/9

75/175 + -6/17 = 9/119
75/175 - -6/17 = 93/119
75/175 * -6/17 = -18/119
75/175 / -6/17 = -17/14

75/175 / 0/1 = 0/1
```

Submitting:

Pack all your files, class files **and** source code, into a **fully runnable** JAR file (we will discuss how to do this in class this week) called **`hw2.jar`**. The main program that the jar file should execute is my `HW2Tester` program (unchanged). I should be able to run the `HW2Tester` main() program from your jar file with the command:

```
  java -jar hw2.jar
```

Submit your jar file via the Canvas submission link for assignment 2.