CS 61B Lab 2
February 1, 2005

Goal: This lab will give you experience with defining and using classes and fields, and with conditionals and recursive functions.

Copy the Lab 2 directory by starting from your home directory and typing:

```
  mkdir lab2
  cd lab2
  cp $master/lab/lab2/* .
```

Getting Started

Read the Fraction.java class into emacs and compile it using C-x C-e, filling in the command javac -g Fraction.java. The program should compile without errors. In a shell window, from your lab2 directory, run the program using "java Fraction". The program should run, although it will print fractions in a non-reduced form, like 12/20.

Part I: Constructors (1 point)

Look at the main method in the Fraction class, which declares and constructs four Fraction objects. Four different constructors are used, each with different arguments.

```
  Fraction f0 = new Fraction();
  Fraction f1 = new Fraction(3);
  Fraction f2 = new Fraction(12, 20);
  Fraction f3 = new Fraction(f2);
```

Look at the implementations of the constructors. The two-argument constructor is straightforward. It assigns the parameters to the numerator and denominator fields. The one-argument constructor uses some new syntax:

```
  this(n, 1);
```

The effect of this statement is to call the two-argument constructor, passing n and 1 as arguments. "this" is a keyword in Java, which normally refers to the object on which a method is invoked. In a constructor, it can be used (as above) to invoke a different constructor.

We could have written the one-argument constructor thusly:

```
  public Fraction(int n) {
    if (n < 0) {
      System.out.println("Fatal error:  Negative numerator.");
      System.exit(0);
    }
    numberOfFractions++;
    numerator = n;
    denominator = 1;
  }
```

Why call the two-argument constructor instead? The reason is one of good software engineering: by having three of the constructors call the fourth, we have reduced duplicate code--namely, the error-checking code and fraction counting code in the first constructor. By reusing code this way, the program is shorter, and more importantly, if we later find a bug in the constructor, we may only need to fix the first constructor to fix all of them. (This principle applies to any methods, not just constructors.) In your own programs, if you find yourself copying several lines of code for reuse, it is usually wise to put the common code into a new shared method.

The no-argument constructor does not use the good style just described. Modify it to call the two-argument constructor. Then, fill in the fourth constructor so that it uses the good style and correctly duplicates the input Fraction (it does neither now). Your TA or lab assistant will ask to see your constructors when you get checked off.

Part II: Using Objects (1 point)

Further on in the main method, there are four lines commented out. Remove the comment markers and fill in the two missing expressions so that sumOfTwo is the sum of f1 and f2, and sumOfThree is the sum of f0, f1, and f2.

Part III: Defining Classes (1 point)

The changeNumerator and fracs methods don’t work. Fix them. You may NOT change their signatures. Each fix should require the addition of just one word. These changes may or may not be in the methods themselves.

Part IV: Conditionals and Recursive Functions (1 point)

The main method prints the Fractions thusly:

```
  System.out.println("The fraction f0 is " + f0.toString());
  System.out.println("The fraction f1 is " + f1);   // toString is implicit
  System.out.println("The fraction f2 is " + f2);
  System.out.println("The fraction f3 is " + f3 + ", which should equal f2");
```

How does Java know what to do when printing the fractions f1, f2, and f3? In the case of f0, we have invoked the toString method, which you should look at.

In the next three lines, we are asking Java to concatenate a Fraction to the end of a String. A Fraction is not a String, so can’t be concatenated directly, but Java cleverly looks for a method called toString to convert each fraction to a string. This is standard in Java: any object can have a toString method, and if it does, that method will be automatically called when appropriate.

As we noted earlier, the toString method prints a Fraction in non-reduced form. Examine the code in the toString method. It is calling another method called gcd that computes the greatest common divisor (GCD) of two positive integers. If this method worked correctly, toString would print Fractions in reduced form; instead, gcd always returns 1. Rewrite the body of gcd so that it is a recursive function that correctly computes the GCD. Recompile and run your program.

Here is a recursive GCD function written in Scheme. a and b must be non-negative.

```
  (define (gcd a b)
    (if (= b 0)
      a
      (gcd b (remainder a b))))
```

The "remainder" operation in Java is called "mod" for "modular arithmetic" and is written "a % b".

```
Check-off
---------
1 point:  Show the TA or lab assistant your last two Fraction constructors.
          Run your program to demonstrate that f2 and f3 are initally equal.
1 point:  Demonstrate that your program correctly computes the two sums of
          fractions.
1 point:  Demonstrate that your program changes f3 to 7/20, and prints the
          correct number of Fraction objects.  Tell the TA or lab assistant
          what two words you had to add to fix these bugs.
1 point:  Demonstrate your program that correctly computes GCDs and fractions
          in reduced form.
```