COMP251 - Week1 - Lab1

Goal: This lab will give you a review to Java Programming.

Getting Started

Download the files for this lab from blackboard. This lab has three parts, you should submit the first part as your lab submission.
Please do the second and third parts, for your learning sake.

Part I: ErrorTests

Download the ErrorTests from blackboard and expand it. ErrorTests directory contains 5 packages. Each one is a separate example. Create one project in Eclipse called ErrorTests. Then create 5 packages (ErrorTest1,...) one for each example. Import the source codes to each package. For each one you have to fix the errors in the code. In some cases you have to add a method, a class, an object,... to fix the error. Make sure all codes compile and run properly

Submission: Save all the examples in one directory (containing 5 sub directories: ErrorTest1,...). Compress it, (please use zip) and submit to blackboard.

Part II: Class Names

Class Names implements some preliminary actions on names and initials. Create an empty project in Eclipse and call it `Names`. Add class: `Names` then compile and run the code.

The program has an error. What is the method (i.e., procedure) that generated an error? What is the line number within the file `Names.java`?

The error is in one of the methods in the String class, which is a standard Java library.

When you think you have found the error, correct it, save the file, recompile it, and execute it to see if the problem is solved.

Part III: Class Fractions

This part has 4 sub-parts (A, B, C and D). Create a project in Eclipse and call it `Fractions`. Add the class `Fractions.java` to this project.
Look at the main method in the Fraction class, which declares and constructs four Fraction objects. Four different constructors are used, each with different parameters.

```
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-parameter constructor is straightforward. It assigns the parameters to the numerator and denominator fields. The constructor with one int parameter uses this syntax:

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

The effect of this statement is to call the two-parameter constructor, passing n and 1 as parameters. "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 constructor from within another constructor.

We could have written the one-parameter 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-parameter 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 might only need to fix the first constructor to fix all of them.

This principle applies to methods in general, not just constructors. In your own programs, if you find yourself copying multiple lines of code for reuse, it is usually wise to put the common code into a new shared method.

A. The no-parameter constructor does not use the good style just described. Modify it to call the two-parameter constructor. Then, fill in the fourth constructor so that it uses the good style and correctly duplicates the input Fraction (it does neither now).

B. 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`.

C. 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.

D. 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; please read the `toString()` code.

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 you concatenate the object to a String. (Actually, *every* object has a toString method, but the default toString isn't particularly enlightening.)

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 pseudocode for a recursive GCD function. a and b must be nonnegative.

```
function gcd(a, b)
  if b = 0
    return a
  else
    return gcd(b, a mod b)
```

Submission

There is no submission for this lab. This lab is a practice for you to review your programming knowledge of Java