C212 Final Exam Review (Part 1/3)

1. Design code to reduce a fraction to its lowest terms. For example 6/8 should be reduced to ¾.
2. Design a class to model fractions and include the code from (1) in its design.
   Recall this is a PDF so you can magnify on the screen as much as you need to see clearly.

```java
class Fraction {
    int num;
    int den;
    Fraction(int n, int d) {
        this.num = n;
        this.den = d;
    }
    public String toString() {
        return this.num + "/" + this.den;
    }
    Fraction add(Fraction other) {
        return new Fraction( this.den * other.num + this.num * other.den ,
                             this.den * other.den );
    }
    Fraction sub(Fraction other) {
        return (this.add( new Fraction(other.num *
                             -1, other.den) )).simplify();
    }
    Fraction simplify() {
        Fraction clone = new Fraction(this.num, this.den);
        clone.reduce();
        return clone;
    }
    void reduce() {
        int m = Math.min(this.num, this.den);
        for (int i = 2; i <= m; i++)
            while (this.num % i == 0 && this.den % i == 0) {
                this.num /= i;
                this.den /= i;
            }
    }
    public static void main(String[] args) {
        Fraction a = new Fraction(1, 4);
        Fraction b = new Fraction(1, 2);
        Fraction c = a.add(b);
        c.reduce();
        System.out.println(a + " + " + b + " = " + c);
        Fraction d = (new Fraction(1, 2));
        Fraction r = d.add(d);
        System.out.println(d + " + " + d + " = " + r + " = " + r.simplify() );
        System.out.println(r);
        Fraction x = new Fraction(1, 2), y = new Fraction(1, 4);
        System.out.println( x.sub(y) );
    }
}
```

3. Write (3 + 4) * 5 with BigDecimals, then discuss and start implementation of a BigDecimal class.
4. Write a program that evaluates simple arithmetic expressions as follows:
   ```java
   java One 3 * 4    returns 12
   java One 2 + 4 * 3 returns 14
   java One (2 + 4) * 3 returns 18
   ```
   So the only requirement is that you separate the tokens on the command line with spaces.
5. Recall the basic Scheme program

```
(define sum
  (lambda (nums)
    (if (null? nums)
        0
      (+ (car nums) (sum (cdr nums)))))

> (sum '(2 5 4 1))
12
> (sum '())
0
```

Implement the data type and the procedure to sum in Java.

6. Write a recursive sort procedure in Scheme and implement it in Java as above.

Reference material from Lecture Seven:

```
import java.math.*;

class Calculation {
    public static void main(String[] args) {
        double initialBalance = 10000;
        double yearlyInterest = 6.0 / 100;
        double monthlyStipend = 50;
        double calculatedBalance = initialBalance * (1 + yearlyInterest / 12) - monthlyStipend;
        double verification = 10000 * (1 + 6.0 / 100 / 12) - 50;
        System.out.println("Consider this: \n  10000 * ( 1 + 6.0 / 100 / 12) - 50 calculates as: " + calculatedBalance + " (instead of " + initialBalance + ")");
        System.out.println("Here it is again: \n  10000 * ( 1 + 6.0 / 100 / 12) - 50 evaluates to: " + verification);
        double value = 4.35;
        System.out.println("Meanwhile: \n  4.35 * 100 = " + (value * 100));
        System.out.print("Now with BigDecimals: \n  ");
        BigDecimal a = new BigDecimal("4.35");
        BigDecimal b = a.multiply(new BigDecimal("100"));
        System.out.println(a + " * 100 is " + b + ");
        BigDecimal balance, yInterest, mStipend;
        balance = new BigDecimal("10000");
        yInterest = new BigDecimal("6.0").divide(new BigDecimal("12"));
        mStipend = new BigDecimal("50");
        BigDecimal cBalance = balance.multiply(yInterest).divide(new BigDecimal("12")).add(new BigDecimal("10000")).subtract(mStipend);
        System.out.println("10000 * ( 1 + 6.0 / 100 / 12) - 50 evaluates to: " + cBalance);
    }
}
```

For the code above since this is a PDF just magnify until it’s clear what it says. Notice how the syntax relates to the program we worked in class, above (Fraction).

---

1 Sort integers in a unidirectional list in ascending order