LAB ASSIGNMENT II

Instructions

- This assignment is a continuation of the B part of first assignment, use the same fraction class and modify it.
- This assignment must be done with ***eclipse IDE***, go through the ***eclipse IDE*** *handout* if you feel difficulties.

Step 4: Defining constructors - Initializing attributes

- The previous lab assignment used setters and getters to initialize and get values, in step4 we will use constructors to set the values.
- Constructors are special methods to initialize an object when it comes into existence.
- While instantiation of a class the constructor gets automatically called.
- During instantiation values of few attributes may be known and can be assigned.
- Constructor(s) bears the same name as that of the class and will not return any value.
- Unlike setters constructors are invoked only at the time of instantiation.

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

public class Fraction {
  private int numerator;
  private int denominator;

  Fraction(int n, int d) {
    this.numerator = n;
    this.denominator = d;
  }

  public void setNumerator(int n) {
    this.numerator = n;
  }

  public void setDenominator(int d) {
    this.denominator = d;
  }

  public int getNumerator() {
    return numerator;
  }

  public int getDenominator() {
    return denominator;
  }

  public void print() {
    System.out.println("The fraction is " + this.numerator + "/" +
this.denominator);
  }}
```

- One may now instantiate and print directly. The effect is the same.

```
Fraction f2 = new Fraction(2,3);
f2.print();
```

- Use of constructor does not make setters redundant. Setters may be necessary for later modification of attributes.
- Implement Driver2.java and use constructor to initialize instead of setters.

Default Constructor

- A default constructor is one that does not take any arguments. It is usually used to assign default values to one or more attributes.

  For example

```
Fraction() {  // default constructor
              this.numerator = 1;
              this.denominator = 1;
           }
```

- You can now check this in the driver code as follows:

```
Fraction f0 = new Fraction();
f0.print();

Output of this code will be 1/1
```

- Modify the print function such that if the denominator is 1, it should not print it i.e. In the above case, the output should be just 1 instead of 1/1
- You can also define another constructor to pass only the numerator. Denominator is set to 1 by default.

  For example

```
Fraction(int n){ // Another constructor
                 this.numerator = n;
                 this.denominator = 1;
                }
```

- There is no harm in having more than one constructor. In fact, this is a feature of Java called polymorphism. Multiple methods or constructors can have the same name provided the number or type of arguments are different.

- It is a healthy practice to define constructors, getters and setters before adding features to the class.

- Defining print methods helps in debugging errors during implementation.

- There are no destructors. JVM takes care of cleaning up the mess (garbage collector).

Step 5: Defining methods - Adding features/functionalities

- The next step is to add the 'real' features to the class that manipulates the attributes.
- GOLDEN RULE TO FOLLOW: IMPLEMENT ONE METHOD FOR EACH FUNCTIONALITY
- Never overload or mix-up multiple functionalities in a single method.
- We define two methods that computes the following:
    - inverse() that inverts the fraction (e.g. 2/3 to 3/2)
    - reduce() that computes the reduced form of the fraction (e.g. 3/12 to 1/4)
- We also define one method to check if the fraction is proper or not.
    - isProper() checks if numerator is smaller than denominator and returns true or false.

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

public class Fraction {
  private int numerator;
  private int denominator;

  /* Constructor section */
  Fraction() {  // default constructor
      this.numerator = 1;
      this.denominator = 1;
  }

  Fraction(int n) { // Another constructor
      this.numerator = n;
      this.denominator = 1;
  }

  Fraction(int n, int d) {
    this.numerator = n;
    this.denominator = d;
  }

  /* Setter section */
  public void setNumerator(int n) {
    this.numerator = n;
  }
  public void setDenominator(int d) {
    this.denominator = d;
  }
```

```
  /* Getter section */
  public int getNumerator() {
    return this.numerator;
  }

  public int getDenominator() {
    return this.denominator;
  }

  /* Print section */
  public void print() {
    System.out.println("The fraction is " + this.numerator + "/" +
this.denominator);

  /* Features section */
  public void inverse() {
    // Swap numerator and denominator. Trivial stuff.
  }

  public void reduce() {
    // Implement your code here. Requires computation.
    // Do it yourself.
  }

  public boolean isProper() {
    // Check and return true or false. Trivial stuff.
  }
}
-----------------------------------------
```

- Add more methods to compute the first 3 multiples, square, square root, etc.
- Create a new driver class "Driver3.java" and call these methods to check their working.

```
Fraction f3 = new Fraction(5,8);
f3.inverse();
if (f3.isProper())
   System.out.println("True");
else
   System.out.println("False");

Fraction f4 = new Fraction(3,12);
f4.reduce();
```

Non-mutable methods

- Sometimes you may want to compute and return the inverse of a fraction rather than modifying the fraction itself.

  This is like saying: Don't invert yourself... Just give me what what your inverse is.

```
public Fraction computeInverse() { // returns a Fraction
  Fraction inv = new Fraction(denominator, numerator);
  return inv;
}
```

Note: The two statements can be replaced by a single statement

```
return ( new Fraction(denominator, numerator) );
```

- From the Driver this method can be called as follows

```
Fraction f5 = new Fraction(5,6);
Fraction f6 = f5.computeInverse();
f5.print();  // f5 remains the same
f6.print();  // f6 is the inverse of f5
```

- The call f5.computeInverse() does not modify f5. Instead it returns a new fraction which is the inverse of f5.
  - You can remove the use of new variable f6 as follows.

```
Fraction f5 = new Fraction(5,6);
f5.print();
f5.computeInverse().print();
```

- In a similar fashion implement non-mutable versions of other operations.

Step 6: Defining Interactions between Objects

- Two or more objects can collaborate with each other to compute some result.
  For example, two fractions can be added to compute the sum.
- This can be accomplished by adding methods which take the other objects as arguments.

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

public class Fraction {
  private int numerator;
  private int denominator;

    ..................
    ..................

  public Fraction add(Fraction frac) {
    // This method can currently handle only fractions with same base
    int numerSum;
    if ( this.denominator == frac.getDenominator() ) {
      numerSum = this.numerator + frac.getNumerator();
      Fraction sum = new Fraction(numerSum, this.denominator);
      return sum;
    }
    else
      return null;
  }
}
-----------------------------------------
```

- The driver code as follows

```
Fraction f1 = new Fraction(2,7);
Fraction f2 = new Fraction(3,7);
Fraction f3 = f1.add(f2);  // same as f2.add(f1)
if (f3 != null)
  f3.print();
else
  System.out.println("Can add only fractions with same base");
```

  - Try giving fractions with different bases as input and check it out.
  - Also, see what happens if you don't have the if ... else condition.
  - In a similar fashion implement methods to subtract, multiply and divide two fractions.
  - Division can be implemented by using inverse and multiply methods defined already.
  - Implement driver to check composite operations like

    ```
    f1.add(f2.inverse()).reduce().multiply(f3).print();
    ```

  - f1.multiply(computeInverse(f1)).print() should output fraction that has same numerator and denominator.

    Play around and deepen your understanding. You need to get comfortable.