

*Project 2*

## Fraction Calculator Application

<b>Course:</b>	INFO3134, S2021
<b>Professor:</b>	<b>Janice Manning and Madhavi Mohan</b>
<b>Due Date:</b>	Friday, August 6, 2021 at 11:59 pm
<b>Student Name:</b>	NOTE: you may work in pairs on this project (optional)

### Fraction Calculator Project:

Create a new project called FractionApp, and include the following classes:

- Fraction.java (an object class)
- FractionCalculator.java (a GUI class)
- LongOperandException (an exception class)
- EmptyOperandException (an exception class)
- DivisionByZeroException (an exception class)

### Description:

Write an application that simulates a Fraction calculator. This app will ask the user to enter integer values for a numerator and denominator of a fraction. It will then present a range of options for the user to perform operations on the fraction, such as displaying it as a decimal, showing its reciprocal value, or reducing it to its lowest terms. As well, it will allow the user to enter a second fraction and then do arithmetic operations on them, such as adding them or multiplying them together.

### Fraction Class:

First, code the Fraction class, which implements the interface Comparable. Review the compareTo() method in the JDK Docs to see the functionality that you will have to implement for this method. Examine the UML diagram below:

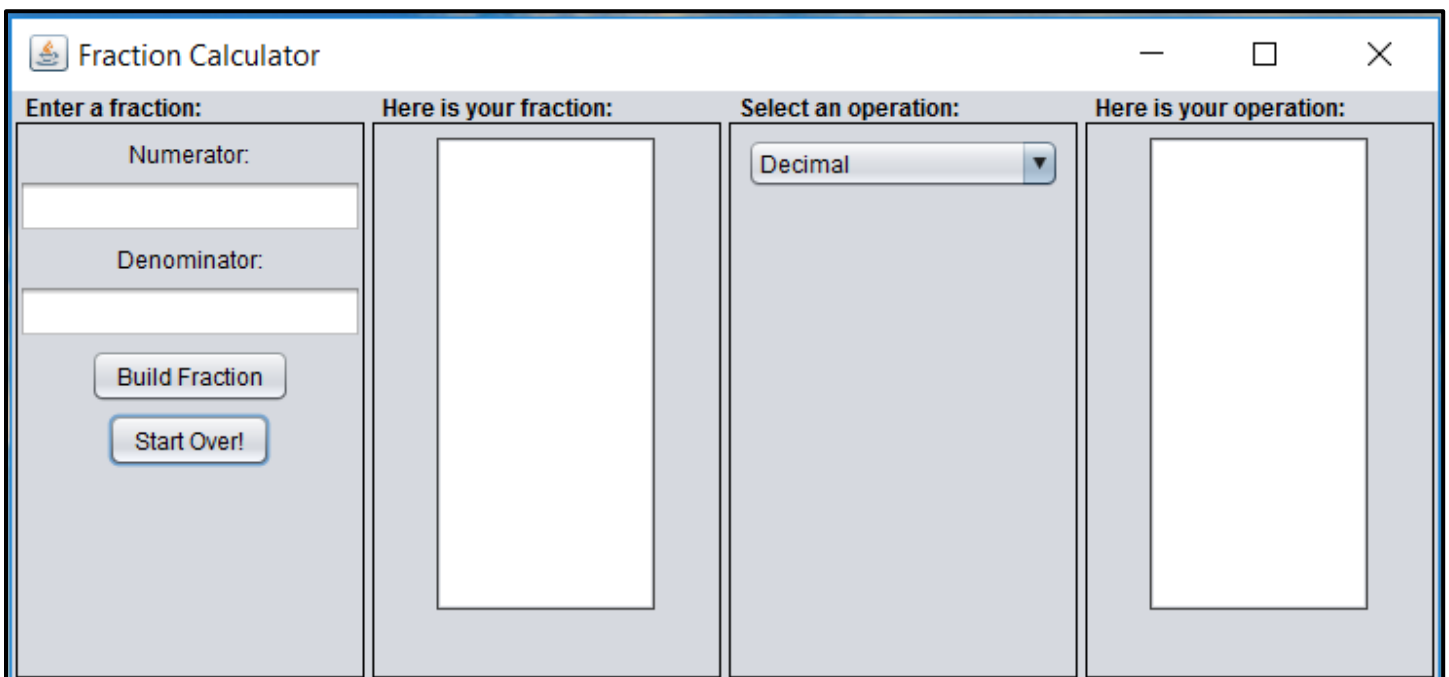
<b>Fraction</b> implements Comparable<Fraction>
- num : int - den : int
+ Fraction() {set num to 1, set den to 1} + Fraction(num : int, den : int)
+ getNum () : int + getDen () : int + setNum (num: int) : void + setDen (den: int) : void  + toDecimal() : double + toReciprocal() : Fraction + add (f : Fraction) : Fraction + multiply(f : Fraction) : Fraction + equals (f : Fraction) : boolean + greaterThan(f : Fraction) : boolean - gcd(den1 : int, den2 : int) : int + lowestTerms() : Fraction  + compareTo(f: Fraction) : int + toString() : String

## Examples of the functionality of the application:

- Here is an example of adding two fractions together:  $\frac{4}{5} + \frac{2}{3} = \frac{12}{15} + \frac{10}{15} = \frac{22}{15}$
- Here is a fraction and its reciprocal:  $\frac{5}{10}$  and  $\frac{10}{5}$
- Here is a fraction as a decimal:  $\frac{1}{2}$  is 0.5
- Here is a fraction in its lowest terms:  $\frac{12}{15}$  in lowest terms is  $\frac{4}{5}$
- Compare two fractions using `greaterThan()` display  $\frac{3}{4}$  is greater than  $\frac{6}{9}$
- Compare two fractions using `equals()` display  $\frac{1}{2}$  is equal to  $\frac{5}{10}$
- A fraction printed using its `toString()`: displays as: "This fraction is  $\frac{3}{5}$ "
- A list of fractions sorted from low to high displays  $\frac{1}{8}$ ,  $\frac{1}{6}$ ,  $\frac{1}{3}$ ,  $\frac{1}{2}$

## FractionCalculator Class:

**NOTE:** The layout of components for the GUI is up to you. The screenshot shown here is just presented as a possible prototype example to get you started. You are encouraged to experiment with different layouts and display options. Do some investigation on principles of good UI layout and apply them.

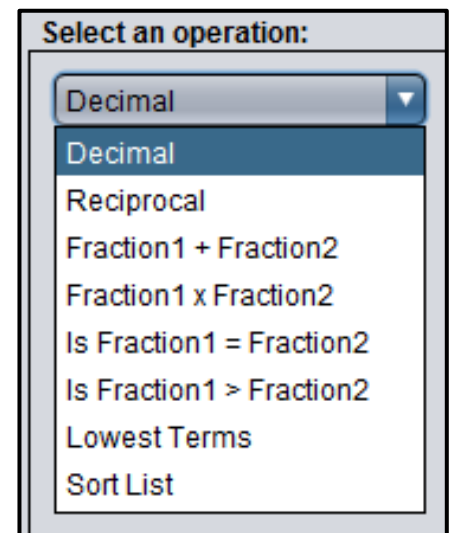


For data input, this prototype uses JTextFields. When the Build Fraction button is pushed, a Fraction object will be instantiated and stored in an ArrayList typed to hold Fraction objects. This ArrayList is declared in the FractionCalculator class. The fraction appears in the list labelled "Here is your fraction". When the user selects the operation from the dropdown list (see the example opposite), the appropriate Fraction method is called and the results are displayed in the list labelled "Here is your operation".

**Provide a limited set of operations that the user can do on the fraction.**

### 1. Unary Operations:

If the user has entered just a single fraction, then he can select the decimal option to view the fraction as a decimal equivalent, or he can select the "reciprocal" option to view the reciprocal value. The user can also select the "lowest terms" option, which would reduce a fraction like  $\frac{2}{4}$  to its lowest term value of  $\frac{1}{2}$ . The result of the operation will appear in the text area labelled here as "Here is your operation:".



## 2. Binary Operations:

If the user has entered two fractions, then he can select one of four binary operations: add the two fractions, multiply the two fractions, compare them for equality, and see if the first fraction is greater than the second fraction.

NOTE 1: if the user has entered more than one fraction into the list, and then chooses a unary operation, *apply it to the last fraction entered*.

NOTE 2: if the user has entered more than two fractions into the list and then chooses a binary operation such as adding them, *apply the method to the last two fractions entered*.

## 3. Sort List:

The method for sorting the list of fractions will be in the GUI calculator class, not in the Fraction class. If this is selected, the fractions should be sorted from lowest value to highest value and then presented in the output area.

Online fraction calculators with good user interface ideas:

<https://www.thecalculatorsite.com/math/fractions-calculator.php>

<http://www.calculator.net/fraction-calculator.html>

[http://www.fractioncalculator-online.com/?gclid=EAlalQobChMIzq-684no2QIV0z2BCh1-JAMdEAAYASAAEgJZFPD\\_BwE](http://www.fractioncalculator-online.com/?gclid=EAlalQobChMIzq-684no2QIV0z2BCh1-JAMdEAAYASAAEgJZFPD_BwE)

## Exception Classes:

Write your own customized exception classes and include them in your submitted code.

1. The **first** exception class should be called *LongOperandException* and will throw an exception if either the numerator or the denominator exceeds the maximum value for the int data type. Its message should appear in a JOptionPane dialog warning box and say something like "Warning: operand entered exceeds int capacity". After the user acknowledges the error, the focus cursor should return to the text box where the offending data value is located and allow the user to modify the entered value.
2. The **second** exception class should be called *EmptyOperandException*. This exception should be thrown if the user has failed to enter a value in either the numerator or denominator text field and then presses the "Build Fraction" button. Again, a JOptionPane dialog warning box should appear and advise the user that either the numerator or denominator box has been left empty. When the user acknowledges the dialog box, the focus should return to the empty text field to allow the user to enter the value.
3. The **third** exception class will be called *DivisionByZeroException*. As its name suggests, if the user enters a fraction with a denominator of zero, this exception should be thrown and caught, and an appropriate JOptionPane warning dialog message should be displayed. After acknowledgement, the focus should return to the offending text field to allow user to make a correction.

## Data Validation:

Your FractionCalculator class code should include some data validation code so that only integer values are allowed to be entered into the input fields. If the user enters a double or non-numeric value, trap this and put up a JOptionPaneDialog warning box advising the user that only integer values are allowed. Return the focus to the field that has the bad data.

## Specification of Functionality:

**NOTE:** The functionalities described here will refer to the components on the prototype on the previous page. Again, the final look and feel of the UI is up to you. The components that you use and how you arrange them is up to you. Just make sure that it meets these functionality requirements.

1. Add functionality to the following components, by implementing actionPerformed methods. You may choose to use an anonymous call to the method or you may choose to use a private inner class.
  - a. For the “Build Fraction” button:
    - Get the numerator and denominator input from the text fields. Validate that the values are integers and then create a Fraction object
      - If either of the text fields is empty, throw the appropriate exception and warn the user and request the appropriate input (either a numerator or a denominator) using a JOptionPane warning dialog box. Return focus to the field with the offending value.
      - If either of the operands exceed the upper limit of the int data type, throw the appropriate exception, and inform the user using a JOptionPane warning dialog box. Return focus to the text field holding the offending value.
      - If the user enters zero as the denominator, throw the appropriate exception, warn the user and using a JOptionPane message box and return focus to the field holding the offending value.
    - Add the Fraction object to the ArrayList list
    - Display the fraction in the Fraction listbox
    - Clear numerator and denominator input
  - b. For the “Start Over!” button:
    - Clear all GUI components
    - Clear all class variables
  - c. For the “Operation” combo box:
    - If a Fraction has not yet been created and added to the list, warn the user and request that a Fraction object be built using a JOptionPane message box, before an operation can be performed. Send the focus back to the numerator input box.
    - Check which combo box operation is selected
    - Call the appropriate Fraction method to perform the operation on the current fraction(s), using the list
      - Be sure your code does not try to perform a binary arithmetic or logical operation on less than two fractions. For example, if only one fraction has been added to the list and the user tries to do an addition, warn the user using a JOptionPane message box and request another fraction
    - Do operations on the most recent entries in the list
      - If the operation selected operates on one fraction but there are multiple fractions on the list, do the operation on the last fraction
      - If the operation selected operates on two fractions, do the operation on the last two fractions on the list; that is, **the second last fraction calls the operation on the last fraction on the list.**
    - Display the result in the Operation text area
    - The lowest terms operation operates on the entire list, and updates the Fractions in the list *and* the Fractions displayed in the Fraction text area
    - The sort operation operates on the entire list, and updates the Fractions in the list *and* the Fractions displayed in both the Fraction and Operation text areas. Your code here will make use of your compareTo() method defined in the Fraction class. Use this method to compare fractions in the list and swap their locations in the list as required. Then display the sorted fractions.
    - Restore the “Decimal” operation to display at the top of the combo box

2. Write an embedded main method using the following code, like so:

```
public static void main (String [] args)
{
    //Create an instance of the frame
    FractionCalculator frame = new FractionCalculator();
} //End of main method
```

### **Bonus Marks Available:**

You can get up to 3 bonus marks on your project if you add a JMenuBar and provide JMenu options that will duplicate the options available on the dropdown list. Each menu item should also have a hot key combination that will allow it to be accessed using keyboard command. The menu should include an "About" item that lists the names of the coders.

### **Submission Requirements:**

1. In File Explorer, navigate to your Project1 folder. Zip the project for submission.
2. Submit your .zip file to the Submissions folder by the deadline indicated on page 1 of this document.

### **Submit your project on time!**

3. Code submissions to the FOL drop box must be made on time! Late projects will be subject to late penalties as follows:
  - up to 24 hours late, 20% deduction
  - over 24 hours late but less than 48 hours late, an additional 30% off
  - over 48 hours late, mark of zero is given.

### **Submit your own work!**

4. It is considered cheating to submit work done by another student (unless it is your partner!) or from another source as your own work. This is called plagiarism. Helping other students cheat by letting them copy your source code files is called abetting plagiarism. Both are considered to be academic offences.
5. **YOU MUST WRITE YOUR OWN CODE!**  
Students are encouraged to share ideas and to work together on practice exercises, and you can help someone else to fix a bug in their code, but any code or documentation prepared for your project must be done by you. Penalties for committing plagiarism, or helping another student plagiarize by sharing source code with them, include a zero grade and an academic offence being filed against the student or students involved. All submissions will be analyzed using plagiarism detection software especially designed to analyze Java code.

## Fraction Calculator Application - Marking Scheme

Description	Available	Awarded
<b>Overall Coding Style</b> - Each class includes: <ul style="list-style-type: none"> <li>A complete documentation header</li> <li>Javadoc style documentation for all methods</li> <li>Adequate commenting within methods, correct indenting, correct use of whitespace</li> </ul>	1 1 1	
<b>Fraction class</b> <ul style="list-style-type: none"> <li>All specified Fraction members are included (<i>as provided with the Fraction demo</i>)</li> <li>compareTo() method is correctly implemented (<i>see next page for a correct solution</i>) <ul style="list-style-type: none"> <li>make this Fraction and the arg Fraction common terms</li> <li>implement Comparable's abstract method <ul style="list-style-type: none"> <li>return 1 if this &gt; f</li> <li>return -1 if this &lt; f</li> <li>return 0 if this == f</li> </ul> </li> </ul> </li> </ul>	1  1 3	
<b>FractionCalculator Class</b> <ul style="list-style-type: none"> <li>GUI components are declared in class scope and initialized in the constructor or in panels</li> <li>An ArrayList&lt;Fraction&gt; has been declared in class scope and initialized in the constructor</li> <li>Use of an appropriate layout and components</li> <li>Components are accurate</li> <li>Numerator and denominator input: <ul style="list-style-type: none"> <li>Data validation is done as specified on both numerator and denominator input</li> <li>Fraction object is created, added to the list, and displayed in txtFraction</li> <li>Text fields are cleared</li> </ul> </li> <li>Unary operations are accurately performed on the last fraction in the list and displayed in txtOperation: <ul style="list-style-type: none"> <li>Decimal, Reciprocal</li> </ul> </li> <li>Binary operations are accurately performed on the last two fractions in the list and displayed in txtOperation: <ul style="list-style-type: none"> <li>Addition, multiplication, equals, greater than</li> </ul> </li> <li>Lowest Terms operation reduces the Fractions and updates: <ul style="list-style-type: none"> <li>list, txtOperation and/or txtFraction</li> </ul> </li> <li>Sort operation sorts the list and updates: <ul style="list-style-type: none"> <li>list, txtOperation and/or txtFraction</li> </ul> </li> <li>Restore the "Decimal" operation in the combo box after each operation</li> <li>Start over clears and resets GUI components, clears class variables</li> </ul>	4  1 2 1  2  4  1  2 1 1	
<b>Exception Classes and Exception Handling</b> <ul style="list-style-type: none"> <li>Each class properly declared and each constructor assigns appropriate message to exception object</li> <li>Appropriate exceptions are thrown, caught, message boxes displayed, input is cleared <ul style="list-style-type: none"> <li>DenominatorOfZeroException should be thrown from Fraction's constructor and setter</li> <li><b>One</b> message box should appear for each type of exception caught</li> </ul> </li> </ul>	/1 /2	
<b>Total Marks</b>	/30	
<b>BONUS MARKS</b> <ul style="list-style-type: none"> <li>JMenuBar added with JMenuItem options for all of the functions listed on the dropdown list.</li> <li>JMenu and each menu item also has a hot key combination that will allow it to be accessed using keyboard command.</li> <li>The menu should include an "About" item that lists the names of the coders.</li> </ul>	+3	
<b>Total Marks Awarded</b>		