# Due \_\_\_\_\_\_\_\_

# AP Computer Science

## Sammamish High School

# Java Project #1

## *Fraction Calculator*

*Students will implement a basic calculator that handles fractions.*

1. **Required Behavior**

* The program should read in fraction equations and print the results in a continuous loop until the user types "quit"
* Input will be either mixed fractions, proper fractions, improper fractions or integers
* Input will be separated by spaces – exactly one space between each fraction and operator.
* The integer and fraction parts of a mixed fraction will be separated by an underscore.
* Negatives are allowed – the negative sign should go immediately before the whole part of the number (with no space in between)
* The operators will be +, -, \*, and /
* The output needs to be in standard mixed fractions, properly reduced (i.e. 1/2 instead of 2/4, 1\_1/4 instead of 5/4)

1. **Sample Execution Log (user input bold and underlined)**

Welcome to the Fraction Calculator!

Enter an expression (or "quit"): **1/2 + 1/3**

5/6

Enter an expression (or "quit"): **1\_1/2 + 1/4**

1\_3/4

Enter an expression (or "quit"): **8/4 + 2**

4

Enter an expression (or "quit"): **-1 \* -1/2**

1/2

Enter an expression (or "quit"): **-11/17 + -1/17**

-12/17

Enter an expression (or "quit"): **1/3 \* 3**

1

Enter an expression (or "quit"): **quit**

Goodbye!

1. **Additional Comments**

No numbers in the fraction will exceed the limits of a Java int (between -2,147,483,648 and 2,147,483,647) after they are multiplied – which means the input will be between -32,768 and 32767.

## Debug Output

Your debug output should print the results from parsing the numbers in the format shown below.

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** |  |
| 1/4 + 1\_1/2 | Val1 whole number: | 0 |
|  | Val1 numerator: | 1 |
|  | Val1 denominator: | 4 |
|  | Operator: | + |
|  | Val2 whole number: | 1 |
|  | Val2 numerator: | 1 |
|  |  |  |
|  |  |  |
|  |  |  |
| 8/4 \* 2 | Val1 whole number: | 0 |
|  | Val1 numerator: | 8 |
|  | Val1 denominator: | 4 |
|  | Operator: | \* |
|  | Val2 whole number: | 2 |
|  | Val2 numerator: | None |
|  | Val2 denominator: | None |
|  |  |  |
| -11/17 + -1/17 | Val1 whole number: | 0 |
|  | Val1 numerator: | -11 |
|  | Val1 denominator: | 17 |
|  | Operator: | + |
|  | Val2 whole number: | 0 |
|  | Val2 numerator: | -1 |
|  | Val2 denominator: | 17 |

1. **Checkpoints**

* Due:\_\_\_\_11/20/18\_\_\_\_\_ Checkpoint 1:Input loop that quits correctly and echoes the parts of the input. Your program **must** store the fractions internally as something other than a string before echoing the fractions.
* *Example*:  
  Can has inputz? **1/4 + 1\_1/2**Fraction 1: 1/4  
  Operation: +  
  Fraction 2: 1\_1/2  
  Can has input? **quit**  
  Goodbye!
* Due\_\_11/28/18(p2), 11/29/18(p1)\_\_: Checkpoint 2: See details in the checklist for checkpoint #2. Parsing the fractions: handle mixed and improper fractions and operations. Complete test case list in comments – 10 to 15 equations that test your code paths that can be copied and pasted.
* Due\_12/5/18(p2), 12/6/18(p1)\_: Checkpoint 3: See details in the checklist for checkpoint #3. Do one operation: fully functional +, -, \*, or / including improper, mixed fractions, and reduction of fractions

1. **Grading Scheme (100 points total)**

* Parses input correctly: 10 points
* Addition: 5 points
* Subtraction: 5 points
* Multiplication: 5 points
* Division: 5 points
* Result reduction: 10 points
* Mixed fraction input: 5 points
* Mixed fraction output: 5 points
* Handles negatives: 10 points
* Checkpoints met: 10 points each
* Comments/style: 10 points

1. **Extra Credit:**

2.5 points for calculators that can handle more than one operation (e.g. 1 + 1 + 1 - 1/2)

2.5 points for correctly mainlining the order of operations with more than two inputs (e.g. 1 - 2 \* 4 returning -7 instead of -4)

2.5 points for handling bad input gracefully (e.g. 1 + + 1/2 does not cause the program to crash)

2.5 points for an infinite precision calculator. This means your calculator can handle very large numbers and still produce the right results. You will need to use the Java BigInteger class. This involves a lot of extra work, so consider it carefully or do it only after you have finished the required parts.

Additional extra credit for other advanced behavior based on complexity and completeness. See Mr Srock with your specification prior to implementation of extra behavior.

Maximum extra credit available 10 points.