AP CS Project

Fraction Calculator

Contents

Project Specifications

For this project, you will program a working fraction calculator!

- Input will be mixed fractions, proper fractions, improper fractions or integers.
- The integer and fraction parts of a mixed fraction will be separated by an underscore.
- The operators will be the 4 basic functions, addition, subtraction, division and multiplication.
- The program should run in a continuous loop until the user types “quit.”
- The output needs to be in standard mixed fractions
- Bonus – simplified standard mixed fraction.

Inputs:

> ***entry1*** *< space>* ***+, -, *, or /*** *< space>* ***entry2***
>
> entry1 or entry2 will be a whole number (ie. 1, 2, 3…), a fraction (ie. 1/4, 2/3, 4/5…) or a mixed fraction (ie. 23_1/2, 2_4/5, 323_2/3…). Note the use of the underscore ‘_’.
>
> The only operators allowed are **+** for addition, **-** for subtraction, ***** to multiply and **/** to divide.

Output:

> The resulting solution to the requested calculation from what the user enters. Example: if the person enters 1/4 + 1_1/2 the program should output 1_3/4

Other Examples:

8/4 + 2	shoul d out put	4
-1 * -1/2	shoul d out put	1/2
-11/18 + -1/18	shoul d out put	-2/3
3/4 * 2_3/5	shoul d out put	1_19/20
3_1/3 / 3/7	shoul d out put	7_7/9
12_2/3 - 4_1/6	shoul d out put	8_1/2

There are many more unlisted inputs that you need to test. When your program is tested for grading other fractions that you may not have thought of will be used to see if your program will stop working or give the wrong answer which will affect your grade on the project. So it is in your interest to think of all the types of test cases that could come up and don’t forget into include testing for negatives.

Additional notes:

- Your final program should include at least 4 methods.

- No numbers in the fraction will exceed the limits of a Java int, which is between -2,147,483,648 and 2,147,483,647.

- Test your projects extensively. Think of all possible cases and try them out. This is really important.

- Check out the functions ***Integer.parseInt***, ***Math.abs***, ***String.equals***, and the ***Scanner*** class, as you may want to use all or some of them. The first two are static functions, which mean you do not need to create objects to use them.

- You need to include internal comments within the code that you create explaining each major function of the program.

- In addition to the code, you need to produce and submit the following documentation:
  - o A step-by-step design document / outline of each portion of code that is submitted, explaining how it works and how it is structured. This document is updated for each checkpoint.

  - o A guide for users, clearly explaining how to operate your program. This may be a separate document or something that appears when you run the program.

Checkpoints

- *Checkpoint 1 – Looper (Loop until “quit” is entered).*
- *Checkpoint 2 - Parser (3 Parse Methods).*
- *Checkpoint 3 - LittleCalc (Operations on Fractions)*
- *Checkpoint 4 - FracCalc (Putting it Together)*
- *Final Submission*

Checkpoint and Final Submission Notes:

With each checkpoint and the final submission – you need to include a design document (outline / pseudo code) of the code you are submitting. This design document should grow as you do each submission

Checkpoint #1 – Looper
(Loop until “quit” is entered).

Create a complete program named ***Looper*** that includes a while statement incorporating a sentinel loop that prompts the user for input and continues to prompt until the user enters ‘quit’. You need to include an output prompt of “Enter something: “. Output the line “Program ended” just before the program stops.

There should be 3 tokens entered each time (Strings separated by a space). You should then output each token separately and then return to prompt again.

This is how your project should look in operation:

```
Enter something: programming is cool
programming
is
cool

Enter something: 2_1/2 + 1/3
2_1/2
+
1/3

Enter something: quit

Program ended
```

Update the design document to include the new procedures for this program.

Include comments in your program to clarify your program.

Checkpoint #2 – Parser

(3 Parse Methods)

Write three separate methods and a complete program named ***Parser*** to contain and test them. Establish a String variable named fraction that will contain a whole number, fraction or mixed fraction to be sent to the methods. You will need to get familiar with the function Integer.parseInt to convert a string to an integer to have an integer returned with each method. Have the results of the methods output by the Parser program.

The function of the 3 methods are as follows:

1) ***parseWhole*** will have the string named fraction as it's single parameter and return the extracted whole number from the fraction variable.
2) ***parseNumerator*** will have the string named fraction as it's single parameter and return the extracted numerator from the string.
3) ***parseDenominator*** will have the string named fraction as it's single parameter and return the extracted denominator from the fraction variable.

Parser must, at a minimum, contain the following statements to run and display the outcome of running the three methods:

```
System.out.println(parseWhole(fraction));
System.out.println(parseNumerator(fraction));
System.out.println(parseDenominator(fraction));
```

If fraction = “123_45/678” the output of Parser should be:

```
123
45
678
```

If fraction = “123” the output of Parser should be:

```
123
0
1
```

If fraction = “45/678” the output of Parser should be:

```
0
45
678
```

Update the design document to include the new procedures for this program.

Include comments in your program to clarify your program.

Checkpoint #3 – LittleCalc

(Operations on Fractions)

Write a complete program named ***LittleCalc*** that uses four integers that represent two fractions and a character that is one of four operators (+, -, * or /). The variable names will be numerator1, denomintor1, numerator2, denominator2 and operator. The fraction variables represent fractions as in the following display:

```
numerator1        numerator2
------------  *  ------------
denominator1      denominator2
```

If your numerator1 is 1 and denominator1 is 2 and numerator2 is 2 and denominator2 is 5 and operator is '*' your output should be:

```
1/2 * 2/5 = 2/10
```

Your program should produce the correct result when using any one of the four operators.

Optional: You may simplify your result (i.e, change 5/4 to 1 and 1/4 or 1_1/4) but this step is not required on this checkpoint. It will be required later in the project.

The design document should describe your procedure to determine the results.

The code will need to be able to handle negatives later in the project, but not required for this checkpoint.

Include comments in your program as needed to clarify steps taken and variables used.

Checkpoint #4 – FracCalc

(Put it Together)

1. Create a complete program named ***FracCalc*** to be used to assemble portions of the programs and methods you have completed on previous checkpoints.

2. The first part of FracCalc should be incorporating the code you created in ***Looper*** to prompt the user to enter the requested calculation (ie. 1/2 + 3_1/4) and to pull out the first fraction, operator and second fraction.

3. Use the three parsing methods you wrote in the ***Parser*** checkpoint to establish the workable numbers of the fraction (whole number, numerator and denominator) and then convert the results of the parsing methods into a simple fraction that you will use in the next step.

4. Use the code you wrote in ***LittleCalc*** to perform the math as requested based on the operator the user entered. Display the results.

5. Loop back to request another calculation (again, incorporating code you wrote in the Looper program).

6. Stop the program when the user enters the word ‘quit’

Update the design document to include the new procedures for this program.

Include comments in your program to clarify your program.

Final Submission

1. Enhance the output of your FracCalc program to simplify the answer using the lowest common denominator.
    a. If you arrived at 45/10 as a result it should be converted to 4_1/2. 3/6 should be converted to 1/2.
2. Make your FracCalc work correctly for negative numbers.
    a. Ie. -4_1/2 * -1/2 should result in 2_1/4
3. Most of the points lost in grading in the past have been missed in not testing enough to find bugs, especially with negative numbers.
4. If your FracCalc is running well after lots of testing and you have time, consider adding extra credit items.
5. Be sure the final submission includes:
    a. Your FracCalc program
    b. Your design document describing how your how your FracCalc program works step by step
    c. Your user guide (either separate document or it displays when your program runs).

    ****** Be sure your name is included inside all of these submitted items ******

Project Grading

Grading Point	**Points**	
Input parsing	10	
Addition & subtraction	10	
Multiplication & division	10	
Loops until requested to quit	6	
Handles mixed fractions	10	
Handles negatives	10	
Coding style including descriptive variable names and reduced code redundancy	12	
Documentation (design document, user guide and internal comments)	12	
4 checkpoints (5 points each)	20	
Total points	------- 100	

Check Point Grading

Grading Point	**Points**
Turned in checkpoint	1.0
Followed specifications / directions correctly	1.0
Code produces accurate results	1.0
Coding style (efficient, low redundancy)	1.0
Design Document	1.0

Total points per check point (rolled up into Project Grading above)	5.0

Extra Credit

- 5 points for simplifying your result
- 3 points for calculators that can take in multiple numbers and operations.
  - Example: 1 + 1 + 1 - 1/2 results in 2_1/2
- 5 points for respecting order of operations.
  - Example: 1 - 2 * 4 results in -7 versus -4
- 2 points for error correction, e.g. if a user entering 1 + + 1/2 will not crash the program and output a user friendly message such as “invalid entry”.

Test Plan

Expression	**Expected Result**
1 + 2	3
1/2 + 2	2_1/2
2/3 + 1/3	1
1_5/8 + 1_1/4	2_7/8
2_3/24 + 1_15/24	3_3/4
2_9/34 + 1_15/34	3_12/17
3_1/3 - 2_1/6	1_1/6
200 - 300	-100
-2 - 1	-3
1/2 - 3/4	-1/4
2_1/3 - 4	-1_2/3
3_1/5 - 2_1/5	1
10 * 10	100
1/2 * 4	2
2_1/5 * 1/2	1_1/10
3_1/3 * 1_1/2	5
2 * 1/7	2/7
1000 / 10	100
3 / 2	1_1/2
2_1/2 / 1/4	10
3 / 4_1/2	2/3
2_1/2 / 5	1/2
10 * 10	100
-2 - -2	0
-3_1/2 * 2_1/2	-8_3/4
-2_1/2 / -2	1_1/4
-2_1/2 + 2_1/2	0