CS1002 Assignment 1 230022876 Tutor: Abd Ardati 19 Oct 2023

Overview

The aim of this assignment was to produce a program that can be used by the user for two different purposes, which can be selected at the start of the program.

The first purpose is to let the user input the number of cups, tablespoons and teaspoons, and the program will output the total amount of cups, tablespoons and teaspoons in one line using the correct format and plurality, such as omitting the measurement and cups for situations where the number of said measurement is 0.

The second section of the program is where the user can input a number of millilitres to be used, and the program will use the measurement to separate this value in the cups, tablespoons and teaspoons required to hold this amount. This then uses the first purpose of the program to display the measurement using all the required units, cups, tablespoons, and teaspoons.

Design

The program is split into three different classes:

- UserInterfaceMain: This is the class that contains the main method and handles all the inputs and outputs that the user interacts with including error handling, creating the necessary object for use, and, depending on the section of code that the user selects to use, calls the necessary methods to produce the desired output.
- BakingMeasure: This is a class that is used to represent the first option or purpose of this program, having the variables cups, tablespoons and teaspoons (all of which are input by the user) and a method called prettyPrint that uses these values to output all of the measurement units in a sentence with the correct punctuation and plurality depending on the values of these.
- MetricVolume: This is the class that uses the input value of millilitres and saves this in a private variable. It then has a method called convert which will take this millilitres value, and convert this into the value of spoons, tablespoons and teaspoons, so altogether it will hold the input value of millilitres.

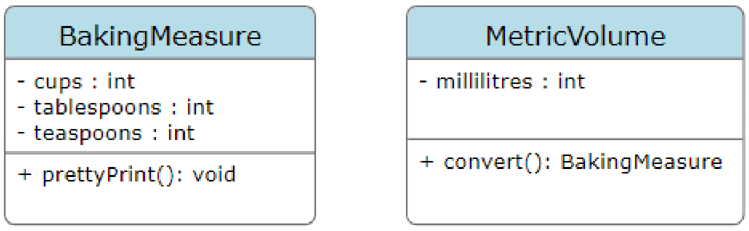


Figure 1: UML Diagram of Necessary Classes[1]

After confirming all the required classes, I created them alongside their required methods and attributes, starting with

The further explain the process in how each individual function works within their respective classes:

- prettyPrint() → BakingMeasure: I was initially planning on creating a conditional for every possibility that could occur to be printed, but there was a much simpler method to use that required far less code, of which I shall explain below:
  - Firstly, I decided to check if all of the values were non-zero (as this would require a punctuation of "," after cups and "and" after tablespoons. If so, there would be

separate switch cases for cups and tablespoons, which would output separate results depending on if there was 1 of the value (singular) or multiple (plural).
    - If the first if statement was not true, then it there would be another selection statement to check if cups is not 0, and if so, the same switch case would occur. However, if there was any case that tablespoons OR teaspoons is non-zero, then an "and" is added to the output statement.
    - After the cups if statement, there is another if statement for tablespoons, to check if this is non-zero, which then uses the same switch case statement for tablespoons as mentioned above. However, after this, if the number of teaspoons is non-zero, then an "and" is added to the output statement.
    - After all this, there is a switch case statement for the teaspoon variable, with cases for 0 (which just prints a new line), 1 (which prints the number of teaspoons as singular) and anything else (which prints the number of teaspoons as plural). This is not reliant on any other conditionals, as there will always be either 0, 1 or many teaspoons, and is accounted for as required.
    - The reason I decided to use switch case statements for this rather than if statements was because the code for each conditional statement was very small, and therefore would be much more readable for other users (as well as being more self-explanatory) with the usage of switch case statements, rather than if statements.
- convert() → MetricVolume: This method was much more straightforward, so does not require any explaining. However, I used the modulus function to get the remainder of the original integer division, so it can be used as the new value of the variable that was being converted from. An example is if there were 20 teaspoons, this would convert to 20 / 3 = 6 tablespoons and 20 % 3 = 2 teaspoons.

The programming for the UserInterfaceMain class was fairly straightforward. The reason why I decided to use one if statement to determine what code to run depending on what the user selected as their option was because the code for each conditional would be considerably large, and therefore would be quite difficult to traverse using a switch case statement. Using an if statement would make this code much more readable and easier to manage.

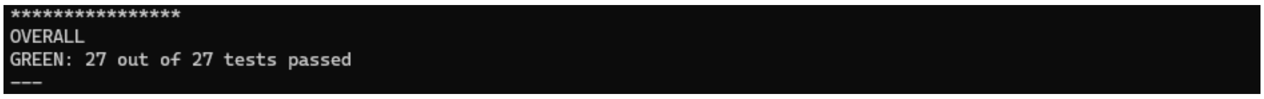
Within this section, I also used embedded if statement, as this would require far less comparisons to occur, when in comparison to having one if statement that contained multiple "if else" comparisons to account for all the different possibilities. In this case, for my current code, when the user has given an invalid input, only 2 comparisons need to be made before determining this, whereas if I were to only have one complex if statement, 4 comparisons would have to be made, which is much more inefficient.

One issue that I did come across in the UserInterfaceMain class was when checking whether the input values was not negative for the user input in the option to pretty print a baking measure. Originally, I wanted to separate this in a separate subroutine that would take the value to check an argument and determine whether this value was non-negative. I also wanted to call this subroutine after every time the user input a value. However, doing this would result in some of the tests in stacscheck not being passed, and therefore had to change this to checking all the inputs after they had all been input. Despite this, I still believe that my original method would have been better programming practice for the situation.

<u>Testing</u>

The first testing that I had done was ensuring that all the automated stacscheck tests were able to pass without any problems.

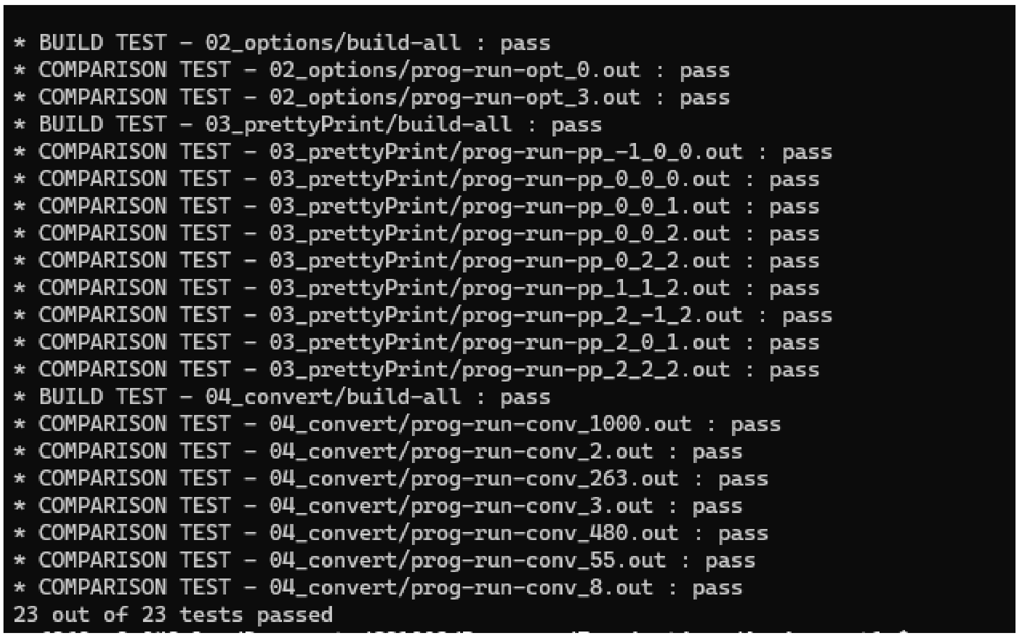
Firstly, I checked whether the first section of these tests passed, which involved making sure that all of the attribute and method names were as required, which all passed as necessary. The results of these tests are below:



```
****************
OVERALL
GREEN: 27 out of 27 tests passed
---
```

Figure 2: Image of Successful StacsCheck Test 1

After this I ensured that all of the tests for the second Stacscheck test had all passed, which tests if the input and output is correct and is as required by the program specifications. The results of the tests are below:



```
* BUILD TEST - 02_options/build-all : pass
* COMPARISON TEST - 02_options/prog-run-opt_0.out : pass
* COMPARISON TEST - 02_options/prog-run-opt_3.out : pass
* BUILD TEST - 03_prettyPrint/build-all : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_-1_0_0.out : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_0_0_0.out : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_0_0_1.out : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_0_0_2.out : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_0_2_2.out : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_1_1_2.out : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_2_-1_2.out : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_2_0_1.out : pass
* COMPARISON TEST - 03_prettyPrint/prog-run-pp_2_2_2.out : pass
* BUILD TEST - 04_convert/build-all : pass
* COMPARISON TEST - 04_convert/prog-run-conv_1000.out : pass
* COMPARISON TEST - 04_convert/prog-run-conv_2.out : pass
* COMPARISON TEST - 04_convert/prog-run-conv_263.out : pass
* COMPARISON TEST - 04_convert/prog-run-conv_3.out : pass
* COMPARISON TEST - 04_convert/prog-run-conv_480.out : pass
* COMPARISON TEST - 04_convert/prog-run-conv_55.out : pass
* COMPARISON TEST - 04_convert/prog-run-conv_8.out : pass
23 out of 23 tests passed
```

Figure 3: Image of Successful StacsCheck Test 2

After this I performed some tests myself, with contained normal inputs, extreme inputs and erroneous inputs. I had written these tests in a table, with the test number, what section of the program it was working on, input values, expected output, final output and test result. There was a total of 16 tests, and the table has been included below:

For the Section of Program, it is as follows:

1. Pretty print a baking measure.
2. Convert metric volume to baking measure.

Test Number	Section Of Program	Type of Test	Input Values	Expected Output	Final Output	Test Result
1	Menu	Normal	1	Enter the number of cups	Enter the number of cups	Pass

2	Menu	Normal	2	Enter the number of millilitres	Enter the number of millilitres	Pass
3	Menu	Erroneous	5	Invalid choice. Goodbye.	Invalid choice. Goodbye.	Pass
4	Section 1	Normal	Cups: 2 Tablespoons: 2 Teaspoons: 2	2 cups, 2 tablespoons and 2 teaspoons	2 cups, 2 tablespoons and 2 teaspoons	Pass
5	Section 1	Extreme	Cups: 1 Tablespoons: 1 Teaspoons: 1	1 cup, 1 tablespoon and 1 teaspoon	1 cup, 1 tablespoon and 1 teaspoon	Pass
6	Section 1	Normal	Cups: 2 Tablespoons: 1 Teaspoons: 0	2 cups and 1 tablespoon	2 cups and 1 tablespoon	Pass
7	Section 1	Extreme	Cups: 0 Tablespoons: 0 Teaspoons: 1	1 teaspoon	1 teaspoon	Pass
8	Section 1	Extreme	Cups: 456 Tablespoons: 23 Teaspoons: 297	456 cups, 23 tablespoons and 297 teaspoons	456 cups, 23 tablespoons and 297 teaspoons	Pass
9	Section 1	Erroneous	Cups: 0 Tablespoons: 0 Teaspoons: 0	Invalid baking measure. At least one unit must be greater than 0.	Invalid baking measure. At least one unit must be greater than 0.	Pass
10	Section 1	Erroneous	Cups: -1 Tablespoons: 3 Teaspoons: 2	Invalid baking measure. Cannot have negative units.	Invalid baking measure. Cannot have negative units.	Pass
11	Section 1	Erroneous	Cups: -1 Tablespoons: -3 Teaspoons: -2	Invalid baking measure. Cannot have negative units.	Invalid baking measure. Cannot have negative units.	Pass
12	Section 2	Normal	Millilitres: 87	6 tablespoons	6 tablespoons	Pass
13	Section 2	Normal	Millilitres: 2340	9 cups, 14 tablespoons and 1 teaspoon	9 cups, 14 tablespoons and 1 teaspoon	Pass
14	Section 2	Extreme	Millilitres: 324257	1370 cups and 9 tablespoons	1370 cups and 9 tablespoons	Pass
15	Section 2	Erroneous	Millilitres: 1	Invalid millilitres. Must be greater than 2.	Invalid millilitres. Must be greater than 2.	Pass
16	Section 2	Erroneous	Millilitres: -5	Invalid millilitres. Must be greater than 2.	Invalid millilitres. Must be greater than 2.	Pass

All of these tests had passed, and therefore I believe the development of the program is completed and works as the specification had required. I believe this testing is sufficient, as it covers multiple possibilities, ranging from normal data to erroneous data, with a valid amount of sample test data, therefore provides an accurate assessment of what it may be used within a real-world environment.

Evaluation

All of the stacscheck tests had passed successfully, and all of my tests that I had performed had also passed as it was required. As well as this, I managed to complete all of the sections of the given specification, and so was able to complete the program as it was required for this assignment.

Conclusion

Overall, this assignment was enjoyable, and definitely allowed me to be able to practice the ability of decomposition, breaking down a problem into smaller steps, planning out a solution to individual sub problems and building it together to produce a full program. It also helped to understand and execute the different types of tests that can be performed to endure the functionality of a program is as it is required.

The only section that I found took longer than preferable was finding an optimal solution in the prettyPrint method, to print out all of the variables to be printed without repeating more lines than was necessarily needed, but this was eventually planned and written to a degree that I believe is satisfactory. With more time, I would have liked to find most possible optimal solution to not have to repeat lines of code for this section, while still having it work exactly as intended.

References

[1] Used https://app.smartdraw.com/ for UML Class Diagram