Object-Oriented Baking Measures Converter in Java

Overview

The primary objective was to create a Java program for the conversion of baking measures and metric volumes. Specifically, we were tasked with developing three classes: `UserInterfaceMain`, `BakingMeasure`, and `MetricVolume`. Program covers all the functionality of the assignment, including 3 classes and required methods.

Code design

Overall, the code complies with the programming style guidelines outlined in the CS Student Handbook[1]

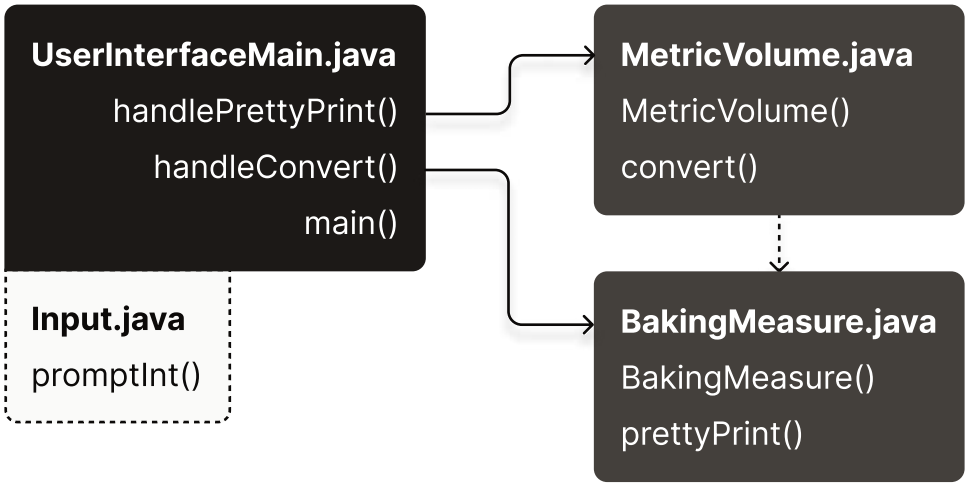

Figure 1. Overall code structure visualisation

UserInterfaceMain.java

I chose to use a `switch` statement for handling choice selection due to its efficiency, simplicity and ease of handling unexpected cases. Additionally, I abstracted the logic for two different cases into separate functions:

```
switch (choice) {
   case 1   ⟶ handlePrettyPrint()
   case 2   ⟶ handleConvert()
   default  ⟶ Print “invalid choice”
}
```

Figure 2. Main switch visualistion

BakingMeasure.java

I introduced the "prettyPrintOneMeasure" helper function to handle plurality for single measures, iterated through measurements, stored results in an ArrayList[2], and used a switch statement for combining values, because there are only 3 completely different options. For larger programs, I would consider dedicated functions for simpler placement of punctuation and conjugates.

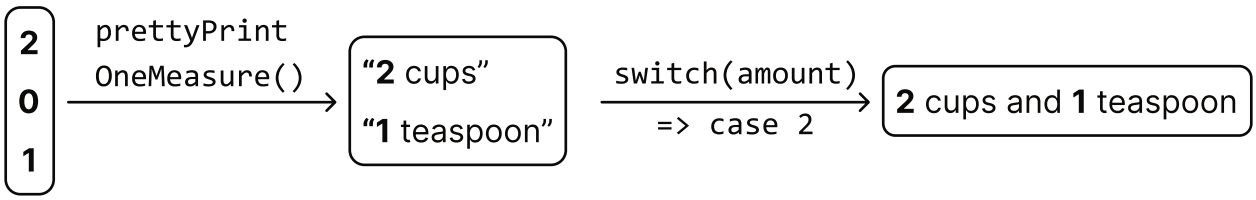

Figure 3. Pretty print logic visualisation

MetricVolume.java

Initially, I attempted a greedy algorithm for conversion, which wasn’t working well with fractions and led to complex conditionals. Switching to an algorithm that is similar to the one in the specification improved code readability and preciseness.

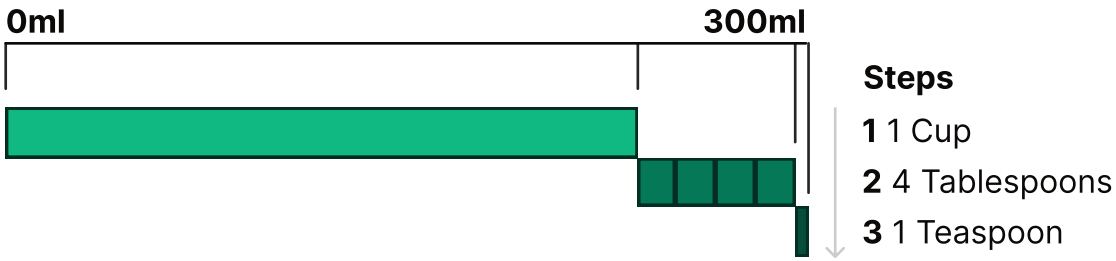

Figure 4. Greedy algorithm visualisation

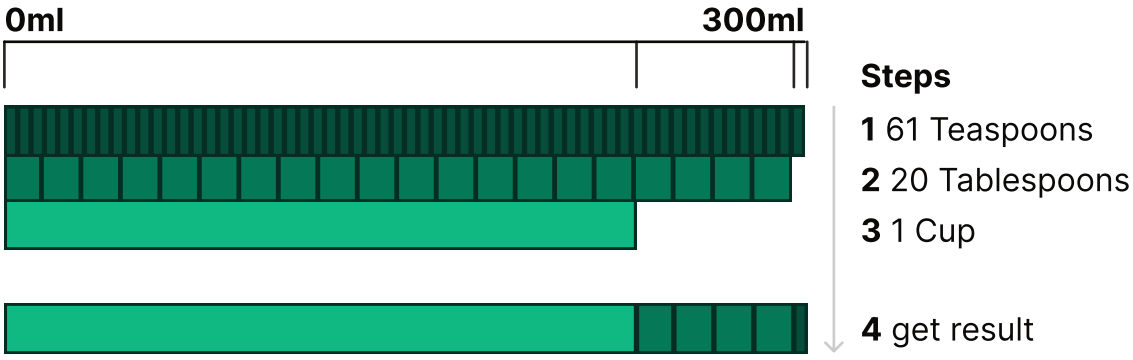

Figure 5. Actually used algorithm visualisation

Additionally, this function's notable feature is the replacement of "magic numbers" with appropriately named constants, enhancing code comprehensibility and maintainability. This change simplifies the code, making it more precise and self-explanatory.

MetricVolume.java > convert() > constants

```
double TEASPOON_CAPACITY_ML = 4.9289d; // teaspoon = 4.9289 ml
int TEASPOONS_IN_TABLESPOONS = 3; // tablespoon = 14.7867 ml
int TABLESPOONS_IN_CUPS = 16; // cup = 236.5872 ml
```

Input.java[3]

The "promptInt()" method abstracts and simplifies error handling for unexpected input values like strings or very large integers, ensuring robust user input handling. This improves code robustness.

Testing

Testing with stacscheck effectively identified and resolved grammar errors and basic logic flaws in the program, aligning it with the specification. While valuable for straightforward issues, it doesn't cover all potential test scenarios.

The summary of 'stacscheck' is detailed below, with all the resulting output saved in both 'output.txt' and 'o.html' files

stacscheck outpput(short)

```
OVERALL
GREEN: 27 out of 27 tests passed
---
23 out of 23 tests passed
```

Further testing

I reviewed various sections of the program thoroughly for some interesting values that might break the program, and the tests that I did are shown on next page.

`Testing main() in UserInterfaceMain.java`

Input	Expected behaviour	Is same in app?
Invalid data		
-100, -1, 3, 53	*Invalid choice. Goodbye.*	✓
string	*Invalid type, ask user to re-enter*	✓
Valid options		
1	*run handlePrettyPrint()*	✓
2	*run handleConvert()*	✓

`Testing prettyPrint() in BakingMeasure.java`

Input	Expected behaviour	Is same in app?
Extreme		
2147483647 *3	*2147483647 cups, 2147483647 tablespoons and 2147483647 teaspoons*	✓
2147483648	*Overflow, ask user to re-enter*	✓
any string	*Invalid type, ask user to re-enter*	✓

`Testing Convert() in MetricVolume.java`

Input	Expected behaviour	Is same in app?
Invalid data		
-100, -1	*Invalid millilitres. Must be greater than 2.*	✓
Exact conversions		
14	*1 tablespoon*	✓
236	*1 cup*	✓
Normal cases		
12345678	*52182 cups, 5 tablespoons and 2 teaspoons*	✓
Extreme cases		
2147483647	*9076922 cups and 6 tablespoons* (to check that fractions don't dissapear)	✓
2147483648	*Overflow, ask user to re-enter*	✓
any string	*Invalid type, ask user to re-enter*	✓

These tests showed that the program wasn't properly working with big numbers and crashed when some unexpected values were entered. All bugs, that I found were fixed.

Evaluation

My implementation effectively fulfills every aspect of the specification. Through thorough testing, I've verified its robustness and reliability.

Conclusion

I completed the program on time, resulting in a functional application and improving my Java programming skills. I also learned how to configure VSCode for remote connections to lab machines, allowing me to work effectively with Linux and stacscheck from my windows laptop.

Writing this report, my first of its kind, was a valuable experience that enhanced my design and copywriting skills. I faced challenges in the code, particularly in integrating fractions into the algorithm, which deepened my understanding of its limitations.

With additional time, I would consider developing a graphical user interface (GUI) to better serve the program's target audience, especially those who may not be familiar with Java and terminal-based applications.

In fact, I had enough time and created a simple website for converting millilitres into baking measurements using Next.js, accessible here:

https://baking-measures.vercel.app/

References

[1] University of St. Andrews. "Programming Style Guidelines." Accessed October 15, 2023. https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/programming-style.html

[2] W3Schools. Java ArrayList. Accessed October 17, 2023. https://www.w3schools.com/java/java_arraylist.asp

[3] Stack Overflow. "How to Prevent a Program from Crashing When a String is Entered Instead of an Integer." Accessed October 17, 2023. https://stackoverflow.com/questions/26375630/hot-to-stop-the-program-from-crashing-after-a-string-is-entered-instead-of-an-in

🇺🇦