CS1002 W05 Practical Report

Samuel Wykes

October 12, 2018

Overview

For this task I was asked to take a certain integer number of kilograms as input and then convert this into imperial stones, pounds and ounces. This information would then be outputted in a readable way with the correct plurality for the units. The program also needed to return an erorr if the user entered a value that wasn't a positive integer.

Design

The first decision I had to make when designing this program was how to decompose the problem into easier to solve subproblems. Using a single method to do the whole program would be code inefficient, confusing for me, and time consuming. I decided it would be sensible to separate the conversion process into its own class which would be called Converter. I could further break down the conversion process by calculating the number of each unit individually and then recombining to give a final answer at the end. I would use conversion constants defined within the Converter class to tell the program how much of each unit corresponded to other untis for example I would have a constant to represent how many pounds are in a stone, ounces in a pound, and kilograms in an ounce. I knew that the Converter objects would also need to store the number of kilograms inputted and the number of stones, pounds and ounces in that number of kilograms so that a suitable output could be provided by the program. This meant I needed integer variables to store all these values. Central to the object would be the convert method which would take an integer kilogram paramater as input set the kilogram variable equal to that parameter and do calculations to set the imperial variables to the appropriate values. Because I needed to output the values in the form "x" stones, "y" pounds and "z" ounces I would need to work out the number of stones first as then I could work out how many pounds remained once the stones had been removed from a running total. Both the stones and pounds variables had to be floored so that you don't say more stones or pounds than were actually inputted. However the number of ounces were to be rounded as there was no smaller subunit being used by the program. I decided to break down the problem even further it would be wise to have three separate methods to convert between units to reduce code reuse. The convert method didn't need to return anything as instead it would store the calculated values within variables in the Converter object and then a separate method called printValues could provide a print out of the stored values.

The printValues method would be where most of the conditionals of the program would go as I have to make sure that the output is formatted correctly. When planning the conditional statements I first focused on plurality. When printing the number of stones, pounds and ounces then I need to print the singular versions (stone, pound, ounce) if the number of that unit stored in the corresponding variables is 1. If the number is greater than that the plural versions need to be printed (stones, pounds, ounces) and if the value is 0 nothing should be printed. This brings up a problem with punctuation as there is a possibility for three different punctuation or the word and positions. The first possible extra word or punctuation position is between where the stones and pounds are being printed. A comma should be placed here only if all three unit values are greater than 0. An and symbol can also be there but only if number stones is greater than 0 and exactly one of number of pounds and number of ounces is greater than 0. The final thing that could be added is an and between the number of pounds and the number of ounces. This and should only be printed if both the number of pounds and number of ounces is greater than 0.

The rest of the problem would be solved in the main method in the W05Practical class. Here instances of EasyIn2 and Converter would be created to take input and do conversion respectively. The main class would also have to print a message to prompt the user to input the number of kilograms they wished to convert.

Testing

Because I decomposed the problem in the design stage of this solution testing the program was relatively straightforward as I could test each individual component separately. This made it easier to isolate and solve bugs. The easiest way to do this style of testing is to test each method individually comparing the expected values to the actual return values. One of the first thing I had to test was whether the individual unit conversions were working. Here I shall provide the testing results for the kilogramsToOunces method. Here I shall be testing a range of inputs and comparing the outputs to what I'd expect by using an online converter and comparing the output of calling the method from the main method with the result given by the online converter for the same arguments.

Input	**Expected Value**	**Returned Value**
1	35.274	32.151239430280036
2	70.5479	64.30247886056007
3	105.822	96.45371829084011
4	141.096	128.60495772112014
5	176.37	160.7561971514002
6	211.644	192.90743658168023
7	246.918	225.05867601196027
72	2539.73	2314.8892389801626
1000	35274	32151.23943028004

From this I deduced that the constant I was using for conversion was incrorect. I changed this value to what it should be (it turns out my google search for conversion of kilogram to ounce gave me an incorrect value.) Once

I reran the program the converted values were the same as those given by the conversion website so I deemed the method to be working. Below are the results of the same tests run with the fixed program.

Input	**Expected Value**	**Returned Value**
1	35.274	35.27399072294044
2	70.5479	70.54798144588088
3	105.822	105.82197216882132
4	141.096	141.09596289176176
5	176.37	176.3699536147022
6	211.644	211.64394433764264
7	246.918	246.9179350605831
72	2539.73	2539.7273320517115
1000	35274	35273.99072294044

Next I needed to test the actual conversion, the way I did this was temporarily making my getters for the different unit variables public and then doing the conversion and printing the variables' values to console. This could then be compared against values obtained using an online converter. In the below table the values in brackets represents (stones, pounds, ounces) outputted.

Input	**Expected Values**	**Returned Values**
1	(0, 2, 3)	(0, 2, 3)
7	(1, 1, 6)	(1, 1, 6)
72	(11, 4, 12)	(11, 4, 11)
1000	(157, 6, 10)	(157, 6, 9)

From this information I guessed that there was an issue with the rounding. This was correct, I was using floor for all three calculations for stones, pounds and ounces. This is appropriate for stones and pounds as you require a remainder to calculate the next largest unit value, however it is more appropriate to round ounces as you want a final answer closest to the true value. This was an easy fix to make as I just changed the code to Math.round instead of casting the variable to int which floors the value. Below is the new result of the tests.

Input	**Expected Values**	**Returned Values**
1	(0, 2, 3)	(0, 2, 3)
7	(1, 1, 6)	(1, 1, 6)
72	(11, 4, 12)	(11, 4, 12)
1000	(157, 6, 10)	(157, 6, 10)

The final individual method I had to test before the program could be tested as a whole was the method printValues() used to print the values stored in the Converter object in a readable format. During the development process I identified three bugs with my initial method. The first bug was that when the initial output message was printed it read "x kilograms in pounds, stones and ounces:" rather than "x kilograms in stones, pounds and ounces:" where x is

the number of kilograms entered. This was an easy fix as I just changed the message being printed.

The second error I encountered was with the plurality in this initial message, I had done if statements for plurality for the stones, pounds and ounces in the second line of the output message but I had forgot to do an if statement to check if only one kilogram had been inputted. This meant if one kilogram was inputted then the message would read "1 kilograms in stones, pounds and ounces:". This was an easy problem to fix as I just had to add an additional if statement to check if the value of the numberOfKilograms variable was 1 and format the text appropriately.

The final bug I encountered was with invalid input. I had forgotten to add a check to see if the user had entered a value less than or equal to 0. This meant that when the autochecker ran, the program failed one of the checks. I fixed this by surrounding the body of the printValue() method body with an if statement checking if the value of numberOfKilograms was less than or equal to 0. If it was the program would print "Invalid input!" and the program would stop.

After all these tests had been completed all that was left to do was test the program as a whole. Because the program had been properly tested in its individual units it worked perfectly when the units that were decomposed at the start were recombined to solve the whole problem.

Questions

1. Explanation of conditional statements.

The conditional statements were not vastly complex for this project and all reside within the printValues() method. The first of which is as so:

```
// print error if value entered was <= 0
        if (numberOfKilograms <= 0) {
            System.out.println("Invalid input!");
        }
        else {
            if (numberOfKilograms == 1) {
                System.out.println("1 kilogram in stones, pounds and
                    ounces is: ");
            }
            else {
                System.out.println(numberOfKilograms + " kilograms in
                    stones, pounds and ounces is: ");
            }

            // create an empty line to append to.
            String outputLine = "";

            // do a check to see if plurality is needed for stone
            if(numberOfStone == 1) outputLine += "1 stone";
            else if(numberOfStone > 1) outputLine += numberOfStone + "
                stones";
            // first check if an "and" is needed
            if(numberOfStone > 0 && (numberOfPounds > 0 ^ numberOfOunces
                > 0)) outputLine += " and ";
```

```
        // next check if a "," is needed
        else if(numberOfStone > 0 && numberOfPounds > 0 &&
            numberOfOunces > 0) outputLine += ", ";

        // do a check to see if plurality is needed for pounds
        if(numberOfPounds == 1) outputLine += "1 pound";
        else if(numberOfPounds > 1) outputLine += numberOfPounds + "
            pounds";
        // check if an "and" is needed
        if(numberOfPounds > 0 && numberOfOunces > 0) outputLine += "
            and ";

        // do a check to see if plurality is needed for ounces
        if(numberOfOunces == 1) outputLine += "1 ounce";
        else if(numberOfOunces > 1) outputLine += numberOfOunces + "
            ounces";

        // print formatted line
        System.out.println(outputLine);
    }
```

I needed a way of checking whether the input was invalid. The specification said to assume the input was an integer so the only condition that was necessary to be met was that the input was a positive integer. Therefore to check this I checked first if the integer was less than or equal to 0. If this is the case then the input is invalid and an error message is printed. The else statement contains the rest of the formatting and works under the assumption that the input is valid which was tested by the first if statement.

Within the body of the first if statement is the rest of the formatting. First of all the program formats the initial line giving the user a handy message to remind them of their input. The only check that needs to be peformed here is testing whether plurality is needed for the output in kilograms, this was a pretty simple if else statement with the condition being that the number of kilograms is equal to 1 or not.

I did the same plurality checks for the other units but instead of a simple if else statement I used else if as I do not want to print anything for these units if the values in the variables are equal to 0. Between each unit value I need to do checks to see if any additional commas or ands need to be added. This was discussed in the design section, the first possible thing to add would be a comma between the stones and pounds values but this only should be added if all the unit values are greater than 0.

The next possible thing to add would be the word "and" between the value for stones and the value for pounds. This should only be added if the number of stones is greater than 0 and exactly one of the values for pounds and ounces are greater than 0, this is represented with an and and xor statement.

The final possible thing that could be added is an and between the value for pounds and ounces. This should only be added if both the number of pounds and ounces are greater than 0.

The way the code runs is such that all the things that are added are appended onto what starts as an empty string. This explains the order of the if statements as I am checking in the appropriate order for things to add.

2. Explanation of number of tests. When testing the program I did tests for a range of inputs. From the smallest possible to a very large input. If the program works for a large range as shown then I can think it likely to work for all inputs (below an extreme large number as floating point representation has its limitations). In an ideal world I would test for every single possible input however this is not practical as I have to manually input each one into an online converter and this would take forever. The fact that my program works for a large range of numbers and passes the autochecker is enough to suggest the program satisfies what the specification tells me to do as it is statistically unlikely that the program is incorrect if it returns valid output for so many inputs.

Evaluation

I was asked to convert a kilogram input into stones, pounds and ounces and output this in a user-friendly, correctly pluralised way. My program does perform this task and performs it using conditionals as specified. Beyond this I was also able to decompose the problem further and separate the solution into more methods which reduces the code reuse, this is good as it made the development process easier and the code more neat and readable. I also made sure to comment the code and use correct naming conventions, further improving the readability and thus maintainability of the code.

Conclusion

During this practical there were a number of things I achieved. Firstly, I was able to successfully decompose the problem into smaller easier to solve subproblems, this problem decomposition made the solution significantly easier to code as I could focus on one easier problem at a time then was able to recombine these solutions to form a fully working program. Secondly, I was able to use multiple classes together to solve the solution, separating the converting process into another class was useful as it made the project much more readabe as the code was a lot less cluttered - separate processes could be kept independent. I was also able to use variables of different scope with certain variables only being used within certain methods and others being available to the whole class. This was useful as I was able to transfer information between the convert() and printValues() methods using the variables within the Converter class. The use of conditionals was also central to the formatting of the output message as I needed to use the correct plurality. Lastly, good coding conventions such as variable naming (using appropriate names for variables and the correct casing conventions), commenting, separating reused chunks of code into variables, and parameterisation.

Extension

Design

For my extension, I decided to do all the tasks in the specification but also design a GUI for the progam rather than have it run through the command line. I would do this using JavaFX and utilising FXML (a markup language) and CSS. I wanted to have it so the user can convert both from kilograms to imperial and vice versa. Not only this but I wanted the user to choose the formatting for the output (i.e the user could choose to have the output in stones and ounces or just ounces.) Similarly to the previous program, I decomposed the problem into smaller, easier to solve subproblems. One of the key differences was the use of another class - Unit. I would use the class unit so that a single convert command could still be used even when the units that are going to be used are ambiguous within the code. This allows for less code reuse as I do not need to write a different method for conversions between every possible pair of units. The unit class would contain the amount of kilograms that goes into one of that unit and a boolean recording whether the user wishes the unit to be used. For example if the user just wants kilograms to ounce converson then the boolean values for the units pounds and stones would be false.

Another thing I need to take into account is how I can have the user interact with the program, I decided it would be wise to have two sides to the program: one side for metric and another for imperial. The user would enter values in either side and then press enter and the program would automatically update the other side. The user would be able to check a box to disable certain imperial units so that they can do different conversions. The way I wanted the program to do these specific conversions was to have an array of units from largest to smallest and then have a remaining kilograms to convert counter. The program would start at the largest enabled unit and find the maximum number of that unit that fits into the remaining kilograms value. It would then subtract that number of kilograms from a running total and move onto the next largest unit.

To convert back from imperial to kilograms the program needs to individually calculate how many kilograms go into each unit and then add them up to form a total. This would then be input into the kilogram text field. To make sure that the correct conversion is occurring (metric -¿ imperial or imperial -¿ metric) then the program needs to be able to detect which text field the user has edited. The way I am planning to do this is to use the text field's onAction method which is called when the user presses enter within any given text field.

One additional thing I need to make sure of is that the user has inputted in the correct format. I will do this by writing a validate function within the main class which takes a string as input and returns a boolean if it matches the format of either an integer or decimal number.

Testing

Evaluation

This program meets all the criteria of the original specification in that it can take in a kilogram input and convert this into stones, pounds, and ounces. It

also fulfills all of the criteria as specified by the extension. Firstly, the user can input in stones, pounds and ounces and this will be converted into kilograms. This is done by the user simply entering values into the imperial units fields and pressing enter. Because of the way the information is outputted - in individual textboxes with labels above informing which unit's being represented - there was no need for an additional class to format the output. The user is also able to choose the output format. They choose the output format by checking the boxes of the units they want to disable and then the program will only convert into the other available units.

Outside of what was mentioned within the specification, I also decided to design a GUI for the user to interact with. This was a good way to practice writing a GUI and made the code a lot more event driven. The GUI components often have a method that is called when the user interacts with them in a certain way.

Conclusion

During the development of this program I was able to show multiple programming skills, first and forermost the use of conditionals was at the heart of my program and I used them throughout the development. For example a conditional is used to validate that what the user entered was correct, following this conditionals had to be used to detect which units the user wants to be converted and do the appropriate calculations.

Another thing I achieved was decomposing the problem into multiple classes each further decomposing the problem with their own methods. For example the class Unit was used so that I could have one method for conversion rather than individual methods for convwerting between every possible pair of units. This reduced code reuse and made development faster.

One thing I found quite difficult was making the GUI look decent as it's something I'm not fantastic at. I also found it difficult to figure out how to allow the user to convert in a given format and disabling certain units from being displayed. Eventually I used a combination of check boxes and disabling the text fields to provide a visual representation and in the calculations I used an array of units from largest to smallest to do a conversion in any given format.

If given more time I would allow the user to create their own units so that more obscure measurements can be used. I would also generalise it further so that the user can convert units that aren't used for measuring mass. As well as this, I'd spend more time on making a more detailed GUI and make it significantly more aesthetically pleasing.