Assignment One Reflection

Tanner Watmough

Student ID: 3554951

COMP 268: Intro to Programming (Java)

In making Assignment 1, I wanted to include all five parts of the assignment within one program. This was a bit difficult as I hadn’t learned about subroutines and calling them yet. To achieve this, I used some online resources, mainly <https://www.youtube.com/watch?v=RiQmRzFRI8k>. This helped me get the basic idea down for how to implement them. Little did I know Unit 4 would cover this in more depth, but I did not know that at the time of starting the assignment. It also allowed me to do a deeper dive on the differences between subroutines, functions, and methods. This was briefly mentioned in one of the chapters from the textbook but I also referred to a Stack Overflow post, referenced here <https://stackoverflow.com/questions/16760460/java-what-is-the-difference-between-subroutine-and-methods>.

My choice to implement all five parts in one file was to try and make it easier for the marker to test the programs. I do realize in hindsight that it does make the file quite long and would likely benefit from separate class files and one main program file, but the implementation is still quite modular in my opinion. I made sure to include test plans and explanations for all the individual parts of the assignment. This should be hopefully clear enough for the marker.

To keep all the parts together, I created a command-line selection menu in the main() routine. This is like how I would implement the Java Help program later. Some of the main issues with the menu were using the switch statements and dealing with invalid options. I did not want to deal with TextIO, as it is not commonly used in the industry from what I read (or rather the scarce search results on it). I will say that TextIO does very well with erroneous entries like alphabet characters vs. numerical characters. To deal with this, I implemented a trick using Strings instead of numbers so entering another character wouldn’t throw an exception error. Apparently, this is actually more efficient according to the official Java documentation found here <https://docs.oracle.com/javase/8/docs/technotes/guides/language/strings-switch.html> but wasn’t possible until Java 7. I originally tried to implement the menu referencing this <https://comptinglearner.com/how-to-create-a-java-console-menu-application/> before running into the exception errors and switching to the String solution.

I also created a separate subroutine for the options menu display, as I wanted to keep the main() routine cleaner for readability. The menu could have been called within the main loop but calling the subroutine is much easier and helps with abstraction in my opinion. This will also be used later for the Java Help Menu problem.

For the multiplication table, I used two nested for loops to deal with the rows and the columns. I then simply multiply the column and row number together (just like a real multiplication table) to get the answer and display it. Because of how I set it up, the first row will be filled out first, followed by the second, and so on which you could likely see on a smaller scale if the multiplication table became large enough for the computer to slow down. You could easily expand or modify the multiplication table size by playing with how many rows and columns there were. You could even start the rows and columns further along if you were more interested in what the multiplication table for the 100s were.

The actual formatting of the table, which was important to the assignment, was done with a printf (print formatted) statement. I experimented with the spacing using the %4d portion. This will print out a decimal, technically, taking up four lines of space. There are no decimals as I did not specify the significant digits and each one results in a whole number (since it’s a multiplication table of whole numbers). After each row has all the columns printed out, it breaks out of the nested for loop and starts a new line. This jumps back to the row for loop at the next number so long as it is less than or equal to the conditional number set. In this case, 12. The row will start at 1 (the 1s row), go until you get 12 rows (the 12s row), and increments by one. You would change these settings to start it later, setting the first to a later number, increase or decrease the size by changing the difference between the row number start and the conditional ending row number. Finally, you can increase the increment by setting the rowNumber equal to the rowNumber plus the steps you wish to increment. You could also reverse the table by decrementing it so long as you reversed the end conditional and starting number or initialized number. Finally, after all columns and rows have been printed, we add one more line of spacing to reprint the menu as it breaks out of the subroutine. This helps with readability.

I had quite a few issues with problem 2 but mostly due to how Scanner deals with errors and thrown exceptions. We start by creating a new scanner and declaring our variables. We then print to the terminal an explanation of the program for the user. Here we could have initialized the values for Fahrenheit, Celsius, and temperature within the declaration, as they are all initially set to 0. A while loop then starts to keep the programming running so long as the Boolean moreConversions is set to true, which it is initialized at as the default value of a Boolean is false. The program then gives a prompt to the user to enter a numerical temperature and then tries to take the next number input by the user. If the user enters an invalid option, we catch the thrown exception and print out an error that a numerical value is required. The system will then print to ask the user to enter a unit. Because the program still has a thrown exception, it prints the question but passes by the user input for the unit as it thinks the user value from the numerical value is also applied to the units. In one case, if the value entered in the numerical temperature is not a numerical value, C, F, Fahrenheit, or Celsius, of any case, it will say it does not understand the units entered, essentially giving two errors, and then asking the user it they want to do another conversion. However, if C, F, Fahrenheit, or Celsius are entered, we will get an output still from the if, else if, else and the value will be assumed to be 0 Celsius or Fahrenheit depending on the unit entered and complete the conversion.

The if, else, else control structure will see if the units variable is equal to Celsius or c, all lowercase, as we convert the string output to lowercase after the user inputs it. If Celsius, we take the temperature variable (the user input which can be Fahrenheit or Celsius, hence why it is not named those variables) and do the appropriate conversion and make it equal to the Fahrenheit value. We reverse this for Fahrenheit. It will then print out a formatted line which will have the degrees in Fahrenheit and the equivalent in Celsius using as many spaces as required and two significant digits. The first number is always temperature and the second always the calculated value, Fahrenheit or Celsius depending on the number entered. Finally, the else will deal with any erroneous entered units and tell the user it didn’t understand their entered units and display it to help them catch any errors they made.

Finally, we take user input for if they want to do more conversions. I once again use a String input, but a Boolean might be more appropriate but also prone to exceptions. In this case I have limited the user to Y or N for yes and no. This is part of the assignment description. If I was to improve, I would likely add more options like the Celsius and Fahrenheit by adding an or operator, ||, that included yes and no. If will then either restart the loop if the response is y and stop it if it’s equal to n, breaking out of the loop. I then have an interesting else loop which tells the user there is an error and that their response must be a capital or lowercase y or n. We then prompt them again, take the next input, and once again repeat the if, else if, else loop. In this case, the else loop, if an invalid response is entered again, will exit to the main menu. This was half a design decision in case the user could not enter a valid entry but also because catching the error exceptions was giving me problems. One of those, it’s a feature, not a bug scenario. If the user either errored out of the loop or said no to more conversions, we print a goodbye message and return to the main program.

Here we utilize some neat tricks, which I’m sure is part of the normal solution, but I was happy I caught in on my own. Here the first column on either side of the table is the same regardless of if it’s Fahrenheit or Celsius. They just increment by 5 regardless and have differing labels and calculations for the second column. Here I messed with spacing and formatting A LOT to get the same look as the example in Assignment 1 for Problem 3. Since the first column starts at -40, and -40 Celsius is equal to -40 Fahrenheit, I set the initial values of columnOne, Celsius, and Fahrenheit all to -40. We then print out the header of the table with the proper spacing and headings as specified in the problem description. We then initialize a for loop counter that continues for 100 values as required, incrementing by one. We use some wonky formatting which will take the values of columnOne, the Celsius, columnOne again, and then Fahrenheit. We start a new line and increment columOne by 5. We then do the calculations for Celsius and Fahrenheit, using columnOne as the value in the conversion calculation to get the respective values. We then print another continue this for as long as the value is less than the conditional value set in the for loop and then print another line of spacing.

The Java Help Menu was implemented similarly to the main menu for the program but specifically needed to accept string values, x, from the start. This is what initially inspired me to figure out the solution for my main menu problem as well. Luckily the string solution makes it very improbable to throw errors with user input and the default case when a non-menu value is entered will display a pseudo-error message to tell the user their entry was invalid. It will then stay within the loop since the switch is nested in a while loop. I also utilized the same principle of having the menu display options in a separate subroutine to hopefully make the program easier to read and use abstraction for the greater good.

The spacing for the menu was experimentation to match the help menu example shown in the assignment description. It also repeats every time an option is selected as otherwise after 5 choices, the user might not know which numbers to choose, or more importantly, how to exit the program. It is a bit repetitive and if I did it over, I would likely try to implement it so every two or three menu choices would print out the menu or implement a tooltip to give you an option to display the menu when you want. The tooltip would essentially be a printed message before the “Selection: “ saying to press 0 or some other value to display the menu again.

As for the actual while loop, I’ve decided to rename the local variable for the userChoice to subchoice. This was done for readability although not specifically required by Java as they are both local to their own subroutines. In this case, the while loop conditional will continue if subchoice is not equal to x, the user option to exit this part of the program as indicated in the assignment requirements. The while loop creates a new Scanner and then displays the helpMenu(). Upon reflection, I do wonder if creating the Scanner each loop iteration, since it is not closed, will keep creating additional scanners and lead to a memory leak? Please let me know if that’s the case as that could be bad. The subroutine then takes the user input, adds a line of spacing, and displays the option. I do have another concern with styling that I’m not sure on. Should paragraphs like those in the middle print statement that span multiple lines be broken into multiple lines in the program, or technically left on the same line? I know some text editors will display the overflow like it’s on different lines without it being the case, so I’m not sure. I know it doesn’t matter to Java as it ignores the carriage returns anyway. I also added a reference for the definition after each description to aid users and to ensure academic integrity. Finally, the program will display a default “error” message if an invalid entry is selected and tell you which values are valid inputs.

The prime numbers problem gave me a lot of trouble mathematically. I looked at <https://beginnersbook.com/2014/01/java-program-to-display-prime-numbers/> after struggling with mathematically and programmatically dealing with finding prime numbers. It was a blind spot from my very dusty math knowledge, as I haven’t taken a math course outside of statistics in nearly ten years. Even then, my statistics course was roughly four or five years ago. In this case, the program was only slightly modified with new variable names to make it easier for me to understand what was going on. We initialize the currentNumber and divisor at 0 but start at 1. The conditional end of the for loop is 10,000, the number required for the assignment description. You could change this to any number you wanted to find all prime numbers up to. The numbers are incremented by one so each one from 1 to 10,000 is each checked once. Within that loop, we initialize a counter at 0.

We then go through a for loop by setting the divisor to the currentNumber which will continue if the divisor is greater than or equal to one, and decrement it by 1. If the currentNumber divided by the divisor is equal to 0, we increment the counter by 1. When the counter is equal to 2, we print out the currentNumber and a space.

Most of the assignment was quite straight forward but I did have some problems with switching from TextIO to the built in Scanner from Java. This was mostly due to the fact users can enter invalid values which will crash the program if the exceptions weren’t caught. I suppose catching all the exceptions is why the TextIO file is quite extensive and long. This mostly gave me issues on the menu problems and the temperature converter as they are the only ones that required input from the user.

My other main limitation from the Assignment was understanding the math problem that is figuring out what is a Prime Number and how to do implement it in a computer algorithm. I could not tell you how to find a prime number mathematically unrelated to programming. Luckily, reading the textbook and the previously mentioned resource from Beginner’s Book website helped get a working algorithm. I explained how I believed the program worked and what modifications could be made in the write up about the assignment, but it highlights an area I’ll need to work on progressing in my programming journey.