Group Five, comprising myself (Link West), Fernando Olivar, and Zion Jassi, are writing a console program to store and provide mathematical assistance with measurement data points. The program will have classes for four different physical measurements: Temperature, Distance/Length, Weight/Mass, and Volume. Within those classes, there will be constructors for creating and storing data points (e.g. Temperature march1520231240pm = new Temperature(64, “F”)). Data points created thusly will then be able to output data conversions through non-static methods, referencing said data points’ values and unit of measurement. Continuing the example with that Temperature data point, one could then enter System.out.println(march1520231240pm.toCelsius), and it would return the converted value of 64 degrees Fahrenheit into Celsius.

As that process is rather straightforward, the complexity of the project presents mostly within the myriad conversions, both intra- and inter-system, that may possibly require calculation. While such conversions don’t pose much of an issue with temperature, as the only units within each system of temperature measurement are Kelvins, degrees Centigrade, and degrees Fahrenheit, with weight/mass, length/distance, and volume, however, there are many different units within both Imperial and Metric systems, such as inches and feet, centimeters and hectometers, cubic centimeters and liters, etc. Each of those possible measurements will have methods, both static and non-static, for being converted one to another. The non-static methods would be useful for converting data points internationally for people who use different systems. The static methods are more informal, and would basically function as a simple calculator, as the methods aren’t tied to actual objects.

An additional degree of complexity we will be adding to the project is in the form of validation methods to check input values for object parameters. When the user creates a data point, the constructor will run a validateUnit() method that compares the entered unit of measurement to an array of acceptable values. If the input matches a value in the array, it will be accepted or converted to a unified abbreviation for that measurement (i.e. “Centigrade”, “Celsius”, and “C” are all acceptable Temperature unit\_of\_measurement values that will be converted to “C”). If the value is not a match, an exception will be thrown stating why, and suggesting that the user then make use of the listUnits() method to view a list of acceptable values for that measurement and system’s units of measurement.

This project *will* assume a degree of expertise on behalf of the user. Rather than spoon-feeding the user prompts in the console and then deciphering said user’s attempts to fulfill those prompts, the expectation will be for a user to be operating in Main, creating and manipulating multiple data points as needed. Assuming this degree of expertise allows for a more direct trouble-shooting process, as we can simply throw an error with the needed information in it to allow the user to correct themselves, when something goes awry in the data entry process. I believe that this approach would mean the difference between having to write a validation method that accepts “Celsius”, “Centigrade”, or “C” compared to one that also accepts “degrees farrenhite” and other such misspellings and extraneous information.