Coursework Part 1 – Reflective Commentary

CSC8001 – Programming and Data Structures

Thomas Harwin (PGT)

Creating the classes for my 'Misty Mountains' Java project within the BlueJ IDE proved to be straightforward with the assistance of the template for initialising variables, creating a constructor for objects within the class and the methods template helped me to understand the basics of object-orientated programming. The '@return' and '@param' functions within the multi-line comments were useful in creating the Java documentation file at the end of the project as well as prompting me throughout the coding process as to how each method worked.

Both my Mountain and Climber classes used a constructor and multiple accessor methods that were simple to write. In my Climber class, I have also included an instance of an array list of the class Mountain called *mountainsClimbed* which is used to record a list of mountains climbed by the climbers. At first, I struggled to grasp the concept of accessing different data types within the ArrayList before realising that each index referenced an object and that I had already created accessor methods for the data types within each object. There are three accessor methods within the Climber class which utilise 'for each' loops to cycle through the indexes within an array list and return specified details about the climber's recorded mountains. Writing these methods made sense to me once I understood how to apply the *getHeight* method written within the Mountain class to the methods I was creating in the Climber class. I found the final method written in the Climber class to be slightly more difficult as it required a second array list to be initialised which would only add mountains with a height greater than a given level.

The Club class is very similar to the Climber class. It uses three similar methods although this time, instead of using 'for each' loops to cycle through an array list of mountains recorded, we are now using the methods defined in the Climber class to cycle through an array list of climbers. I felt that coding the Club class was easier having already had practice writing the 'for each' loops within the Climber class.

Finally, the ClubStats class features the main method of the program, which takes user input using a 'scanner'. I have created a menu that prints onto the console and provides the user with a number of options. By using a 'switch' statement, the number that the user inputs will be read by the scanner and will then take the user into the corresponding switch case. I chose a switch statement instead of 'if / else' statements as having a default case helps avoid any data entry types which are illegal and it is easier to read when there are a large number of cases. There is also an option to exit the program which uses a 'while' statement to continue the switch statement unless the input matches that needed to exit the program.

Looking over the program, there are a number of improvements that I feel could be made. Firstly, the main method defines just one instance of club as per the specification although it could in fact be used to create multiple instances of 'clubs' if needed whereby separate collections of climbers could be recorded. Another potential flaw of the program is that once the user has entered a case in the switch statement, if an incorrect data type is entered the program fails. Through my own reading, this could be countered with a 'try' and 'catch' block that would handle any exceptions to the expected data input and print an error to the user's screen. Finally, I have included an extra option within the main method to print a list of all the climbers in the club to the console. Whilst this wasn't mentioned in the specification, I felt it was important to include should the user wish to keep track of all the climbers they had entered into the club array list. To improve on this, a further sub-menu option could also be included, which would allow you to list the mountains recorded against each climber.