Benjamin Le
19798734

## Justify your menu implementation decisions

I decided that the code for my menu would look much cleaner and would be more efficient if I were to implement a case statement as opposed to having a bunch of if else statements. The user is prompted to pick options between 1 and 7, and so having a case for each option would mean that whichever integer the user would enter would call a certain submodule depending on their choice. I put this entire case statement in a do while loop so that the menu will continue to loop until the user inputs the number 7 which is to exit out of the program. I decided to use a do while instead of a while loop or a for loop because I wanted the menu to be looped through at least one or more times and I did not want the menu to loop through a specific number of times, so this was the loop that made sense to use.

I also created a sub menu which prompts the user to create a submarine or a fighter jet. The user is asked to enter the serial number, year, cylinders and fuel outside of the if else statement because these are the traits that both ships have in common and also helps reduce redundant code, as having both if else statements asking for the same info is pointless. Depending on the type of ship the user wants to ask, the if else statement will also ask the user to fill in other bits of information including the hull, max depth, wing span and ordnance. The reason why I did this was so that the user would not need to enter in details that the ship didn’t require.

## Justify your approach to data validation

In my UserInterface class, I created 3 submodules which dealt with user integer input, real number input and string inputs. For the integer and real number inputs, I decided to import a prompt in the form of a string, a minimum value as an integer and a maximum value as an integer. Reason behind this is because it allows these submodules to be reusable and not hardcoded to only work just this one time.

I created private validation submodules so that other users won’t be able to change the values which were set in the submodules. These validation submodules are then used in my alternate constructor as a check list before created. The reason why I chose to do this is because the object can’t be created until all the class fields that the object requires are valid.

The most difficult submodule I had to validate was the serial number. Originally I did not completely finish the previous variants of the serial number before it being turned into a string, but I figured that by creating an array of Strings to store the 2 different parts of the serial number, it would make validating both individual parts a lot easier. I had to use the split method to split the strings into two separate strings and converted them into integers. I then validate each individual component which checks to see if the strings meet their conditions.

## Justify your design decisions in regards to what functionality was placed in the container classes and what functionality was placed elsewhere

The included functionality of each model class was chosen due to the convenience and ease of access of the data inside the class. For my submarine class, it contains all the specific and necessary information that a submarine would have, and not bits and pieces of other information that belong to other classes. The super class Ship contains the class fields which both submarine and fighter jets have in common – in this case the serial number, year, cylinders and fuel. Fighter Jet class would only contain fighter jet specific submodules which would only be relevant to the fighter jet. Example is that the Submarine class does not require details about wing span and ordnance, hence it is not included in the Submarine class at all. Wing span and ordnance is specific to fighter jets and so submarines will not have those class fields within their respective class.

**Discuss any complications inheritance introduced in your design. If it did not introduce complications then discuss how it simplified your design**

The introduction of inheritance simplified my design, as it allowed me to get rid of redundant code, since submarines and fighter jets had serial number, year, cylinders and fuel class fields in common. I found the inheritance prac to be one of the "easier" pracs as it mainly involved removing redundant code and refactoring previous code to better suit the super class.

**Discuss and justify any down casting present in your code**

The only example of down casting present in my code was used during the equals method.

**Discuss any challenges you had in your implementation/design**

Overall, I found the assignment to be tough because I had no previous coding experience and was trying to learn what was taught in the lectures and then implementing them into the pracs. I found that I was slightly behind every week and would need to try catch up weekly. The hardest implementation in my code would have been finding out how to validate serial number properly, as I wasn't sure how you could properly validate two different components of a string to meet a certain criteria. I left the proper validation of the serial number off for a while.

FileManager was also one of the tougher implementations of my program. I was unable to properly use the save and load ships feature in my menu due to a lack of in depth File IO understanding. I originally had a for loop looping from ii to the length of the shipArray and didn't know why I kept getting NullPointerException errors until I realised I had to loop through the shipCount instead.