# Assignment

## Due: 27 May 2019 at 8am

## Weight: 20% of the unit mark.

I organised all the menu operations to be found in the UserInterface class. I decided to implement sub-menus for many of the modules within this class, the first of which occurs in the addShip method. I implemented a sub-menu in this method so that it was easy for the user to select which ship they would like to add and easy for the program to split off into the appropriate module of inSub, to enter a submarine, or InFj, to enter a fighter jet. Sub-menus again occur in the inSub and inputEngine methods as user input selections for the material of the submarines hull and for the type of fuel the engine runs on. All these menu structures use a do-while loop as they must always be executed at least once, even if the first option selected is to exit or return to the main menu. Switch-cases are then used so that the selected method can be operated, and the program is organised neatly.

I implemented certain validation submodules in certain classes depending on what class fields were present in that class; for example, in the Submarine class I had the validate submodules for the hull and maxDepth class fields. By making use of the super class Ship I was able to take what common validations occurred in both the Submarine and FighterJet classes and use them in the super class instead, thus trimming down on how many validation submodules both the Submarine and FighterJet class needed. The only exception to the super classes good work was that I decided to keep all the validation submodules in the UserInterface class for my own piece of mind as the inSub and inFj modules rely heavily on all the validation submodules to validate if each user input is, well, valid. The validation submodules main role throughout the code was to obviously validate any data that needed validating, this was applied to each class field setter which were then used in the Submarine and FighterJet alternate constructors. Originally each alternate constructor was validating its own class field data that was also getting validation from its setters, I quickly changed that so that only the setters validated for the Submarine and FighterJet alternate constructors. This change meant I did not repeat the validation logic for not apparent reason.

As already stated, I decided that all user interactions with the program would occur in the UserInterface class. The modules of the UserInterface class are the longest of the entire program but each module completed its one specified task in a semi-well-structured manor; this allowed me to understand each step throughout each method. The UserInterface classes functionality relies on the calling modules within the ShipStorage class and the construction of Submarine, FighterJet or Ship objuects. The ShipStorage class mainly consists of operations to an array of Ship objects. These ship objects are partly created by the super class Ship and either the Submarine or FighterJet class. I decided to implement my program like this so that the methods of each of the classes cascades into the next necessary class; essentially meaning the functionality of each class relies on what it inherits from. The best example of this would possibly be the destinationCheck method: a user input for distance begins its journey in the UserInterface class in the submodule destCheck, I implemented an if-statement to insured that the distance was a positive number; from there the distance is imported to the destinastionCheck method of the ShipStorage class, within this method the distance and the cylinders, aquired through calling an appropriate getCylinders method within Submarine or FighterJet, are used to determine the time taken by calling the appropriate travelCalc method of the appropriate Ship object. Down casting is a major part of how the Ship, Submarine and FighterJet classes function together. Both the Submarine and FighterJet classes are extensions of the super class Ship. This down casting leads to such functionality as being able to call different toString methods for each Ship object stored.

The biggest challenge I faced whilst designing and implementing my code was that of time constraints. I fell ill toward the final week of the assignment and doing so fell behind in my work. These ever-present time constraints were felt worst in the late hours before submission when lingering bugs throughout the code never seemed to fade. The completion of such a large program in the time we had proved difficult, leading to its completion being prioritised over the fixing of some small faults.