1.0 Justify your menu implementation decision:

My menu implementation in UI.java consisted of menu options whereby the user inputted a number to select their options. This was made in this manner because I didn't want the user to be typing out long strings of texts or trying to find a character on the keyboard to select their desired option as it's easier to find numbers than characters on a keyboard as all the possible number options belong on one row on the keyboard and characters are scared across the keyboard. The intent of the programme is to be extremely user-friendly and easier to use, using numbers is a clear-cut way to ensure the user can select their menu as easily as possible as numbers eliminate the mistake of missing typing or inputting it in the wrong casing of a characters, therefore, allowing the user to quickly to select their options additionally, it greatly simplifies the algorithms of the menu. The menus follow this simple formula of a case statements encased in a do-while loop. The use of do-while loops is because we don't know how many times the user wants to use the menu, hence the for loop cannot be used as it needs to know how many times to loop. Furthermore, the programme needs to at least show the menu and its options to the user at least once, hence the while loop cannot be used as it doesn't need to loop at all, thus making the do while loop the perfect candidate for the desired use. The case statements are Implemented because the menu options which can be selected are ordinal selections as all combinations can be listed hence, making the use of switch statements the most applicable. On the contrary, the use of if-else statements could've been implemented, although they're more suitable for non-ordinal selections hence, options which cannot be listed easily i.e. checking if a number is between a specific range of numbers or checking boolean expressions.

2.0 Justify your approach to data validation

My approach to data validation is to validate variables each variable/object in their own method whereby an exception will be thrown. The exception is to be thrown in the validate method to make the thrown exception more specific to what is been validated and to increase the cohesion between the submodules. Therefore, when multiple validations are required such as the alternate constructor, specific errors can be thrown when something goes wrong thus making it easier to debug the algorithm and it keeps it relatively short. Additionally, in the case where the validate methods had to be used multiple times it eliminated redundancy as I have I to only throw the exception in one place, and not in every instance I use it. In my UI.java I didn't have the copied validation menu's throw anything. I simply had them return either true or false. This was done because the errors made by the user can be easily controlled through the use of If-else statements and do-while loops. Additionally, we don't want the programme to crash when the user makes a mistake, we simply just want to keep re-prompting the user until they input a correct input. Although for the classes of submarine, ship, fighter jet, ship storage, and engine it was imperative for them to throw exceptions as the input for those class must be correct, otherwise, the programme becomes non-functional. For the methods IsSame, they are two versions, one for validating equality of Strings and the other for validating the equality of real numbers. The validation of equality of real numbers must be validated in a specific tolerance because for real java sees numbers with different decimal places (I.e. 12.002 and 12.0 or 12.00000001 and 12.000) as totally different numbers hence we need to compare them in a given tolerance. The tolerance I choose was 0.001 because we as stated in the assignment, it's stated that all numbers must be displayed to two decimal places, hence there's no point for checking any decimal places after this. The validation of equality of strings was made because the assignment specification sheet doesn't want to take into account casing,

therefore it's imperative to make a module which converts the input to upper cases and then compares the equity of strings, as java's own equals method takes into account the casing of strings.

3.0 Justify your design decisions in regards to what functionality was placed in the container classes. And what functionality was placed elsewhere:

The functionality of getting information from the user, validating the user's input, and displaying messages to the user was placed inside the user interface, this was done because that it the user interfaces responsibility. This follows suit for other classes made i.e. anything to do with fighter jets went into the fighter jet class, anything to do with submarines went into the submarine class, etc. This is done because these are the standard coding practice when it comes to constructing classes.

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

In my experience inheritance simplified my algorithms, it made the algorithms less redundant, and it made it easier to change the common code between the fighter jet and submarine class. Previously where the common algorithms of the serial number and commission year both belonged in a fighter jet and submarine it made it more difficult to alter these algorithms, as you'll have to go in both documents make the necessary changes, and compile both files to check that you haven't made any syntax mistakes. With inheritance, it made it a lot simpler to change the common code between the subclasses of a fighter jet and submarine as you had to only change the superclass, and check for errors in one file. Moreover, inheritance made the algorithms if fighter jet and submarine more digestible, as you're not reading the repeated code, you're only reading code which is specific to that given class. Furthermore, therefore, it made the subclasses algorithms smaller in size. Also, talk about the toString arrays and shit

5.0 Discuss and justify any down casting present in our code:

I downcasted the class fields serialNum, year, and engine to the subclasses of fighter jet and submarine. This is done because these class fields needed the same class fields, additionally, it's to reduce redundant code thus making it easier to read and edit the code later.

6.0 Discuss any challenges you had in implementing your design:

The major challenges I had in implementing my design is the constant changing of the pseudo code. In my experience, once I have converted my pseudo code to java I would find flaws in my code given by the java compiler. I would fix the mistakes and get the algorithms fully functional although it was time-consuming matching my pseudo to my java as time is something, we don't have an abundance of. Additionally, another challenge I had was the time it took to fix your code and ensure that it works properly, I am grateful that the Mark and Amy designed the final workshops to work through your assignment, otherwise I would've grossly underestimated the time it took to complete the assignment.