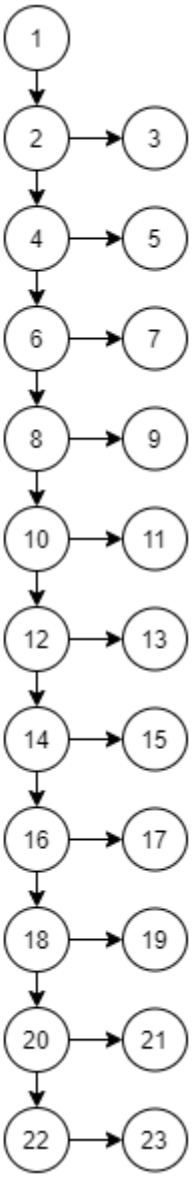
Name: Morad E.

OOSE - Assessed Exercise 1

1.

```
1  private void initialiseVehicle(String vehicleName) {
2      if(vehicleName.equals("Boat")) {
3          vehicle = new Boat("Apollo ");
       }
4      else if(vehicleName.equals("Ship")) {
5          vehicle = new Ship("Cruizz");
       }
6      else if(vehicleName.equals("Truck")) {
7          vehicle = new Truck("Ford F-650");
       }
8      else if(vehicleName.equals("Motorcycle")) {
9          vehicle = new Motorcycle("Suzuki");
       }
10     else if(vehicleName.equals("Bus")) {
11         vehicle = new Bus("Aero");
       }
12     else if(vehicleName.equals("Car")) {
13         vehicle = new Car("BMW");
       }
14     else if(vehicleName.equals("Bicycle")) {
15         vehicle = new Bicycle("A-bike");
       }
16     else if(vehicleName.equals("Helicopter")) {
17         vehicle = new Helicopter("Eurocopter");
       }
18     else if(vehicleName.equals("Airplane")) {
19         vehicle = new Airplane("BA");
       }
20     else if(vehicleName.equals("Tram")) {
21         vehicle = new Tram("EdinburghTram");
       }
22     else if(vehicleName.equals("Train")) {
23         vehicle = new Train("Virgin",4);
       }
   }
```

Q1



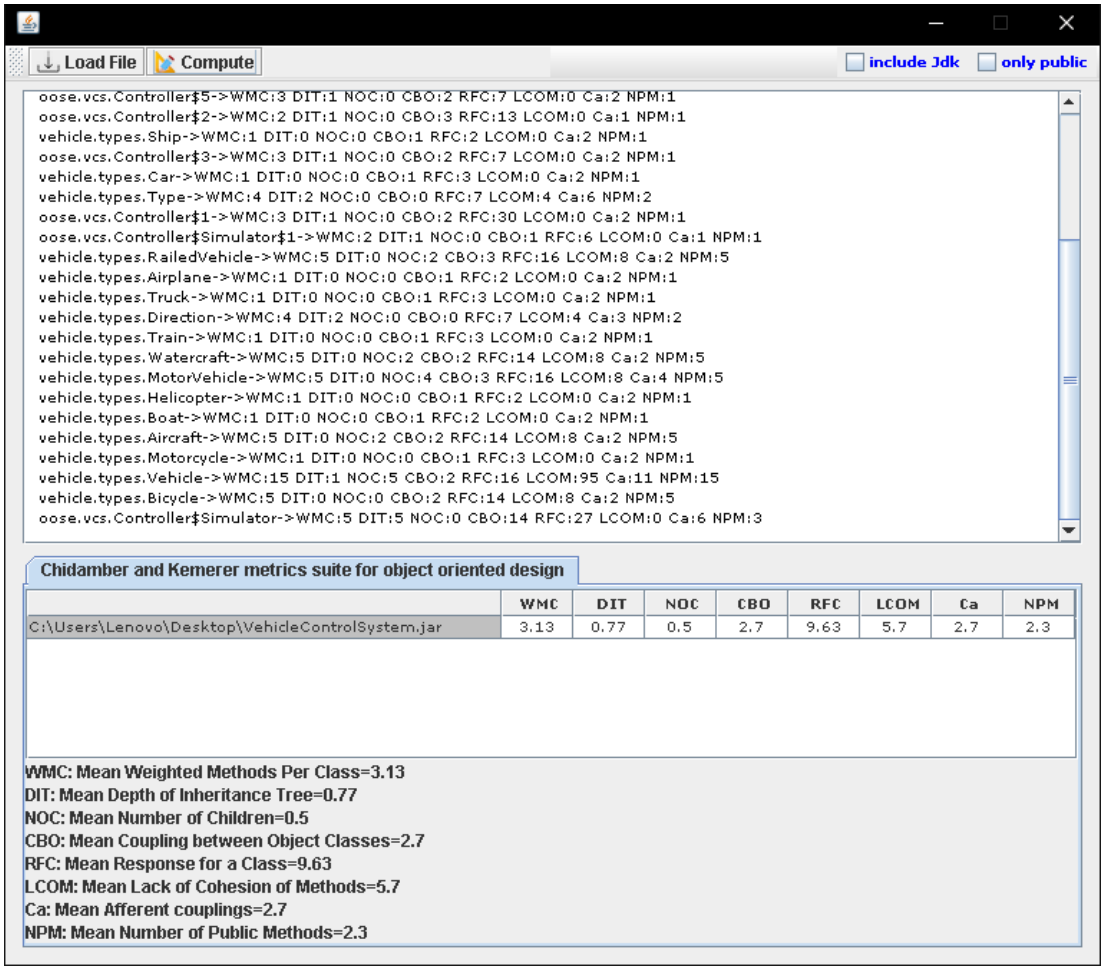
2. One metric which can be used to measure the desired quality factors is CBO: Coupling between Object classes. As this metric looks at the number of classes to which a class is coupled, taking into consideration that reaching a lower CBO value (in other words having less coupling) will result in improved modularity and enhance system encapsulation, it is thus justifiable to make use of this metric. Furthermore, a lower CBO value will effectively assist in reducing complexity in the Vehicle Control System.

Another metric to test against is the Weighted methods per class (WMC). As it is the sum of the complexity of the methods in a class, and since it helps to indicate the amount of effort and time which will be required to develop and maintain the class, achieving a lower WMC value should hopefully serve in indicating that the quality factor of complexity in the Vehicle Control System has reduced.

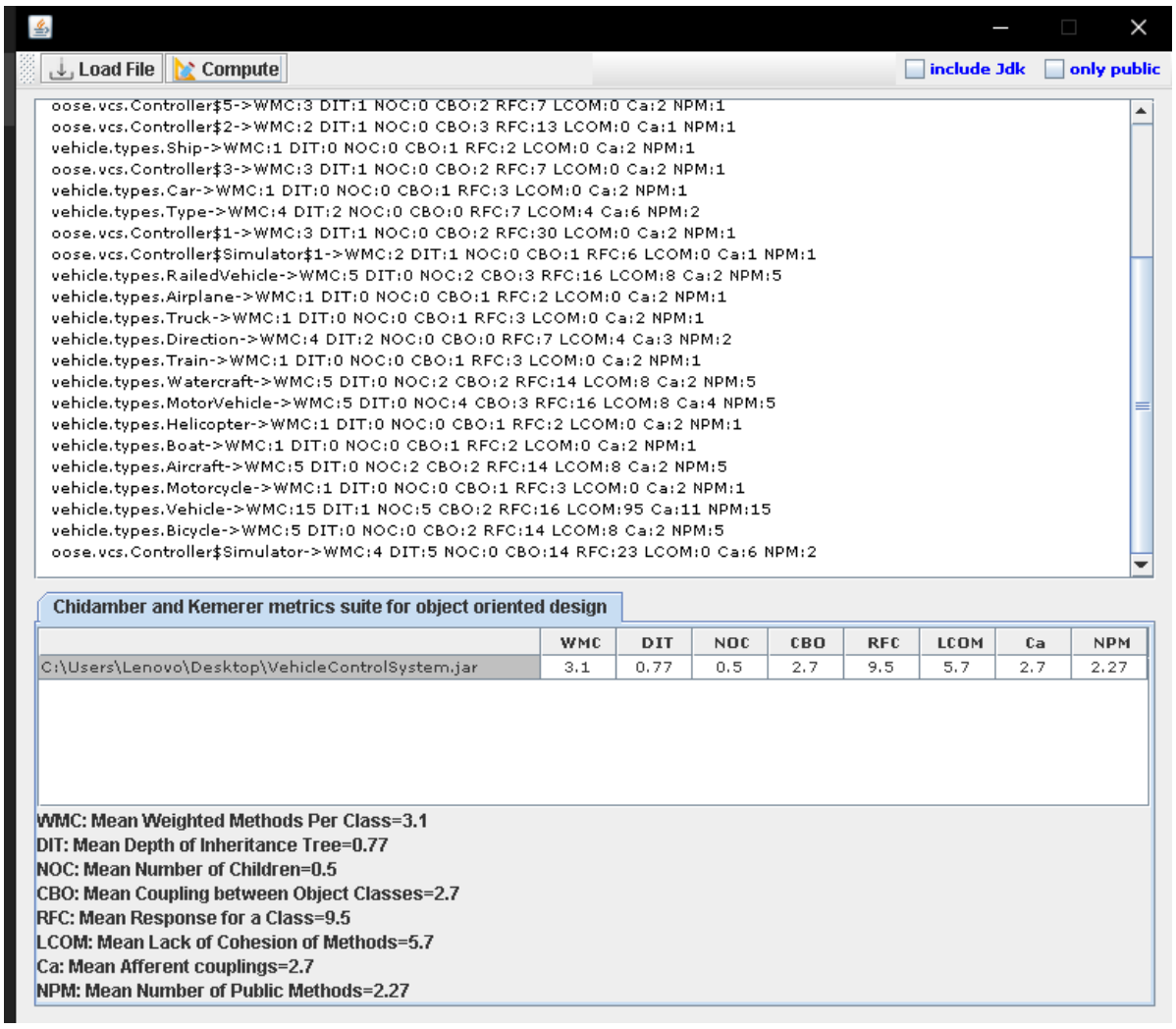
The final metric to work by is the Lack of Cohesion metric (LCOM). As it functions in measuring the tightness of the code, aiding in pointing out areas with poorer quality design, in the case of Vehicle Control System, it can essentially provide insight on the quality factors of complexity and encapsulation which are being measured for in Vehicle Control System.

3. So as to enhance the quality factors, the most important class to be refactored in VCS should be the Controller class. Contrasting with the other various vehicle classes, the Controller class shows very clearly many parts in the code where various poor design decisions have been made. To highlight, there are methods, of which, some are extensively long without needing to be, some attempt to do more than they should be (and this could be noticed specifically in the GUI-simulation overlapping part in the code); to add, it appears there is a lot of code which does not follow with the DRY principle: excessive repetiton of code blocks can be noted throughout within the class. Without needing to be said, the Controller class is complex, and neither well encapsulated nor modular enough. Hence, choosing to refactor this class could better enhance the desired quality factors being sought for.

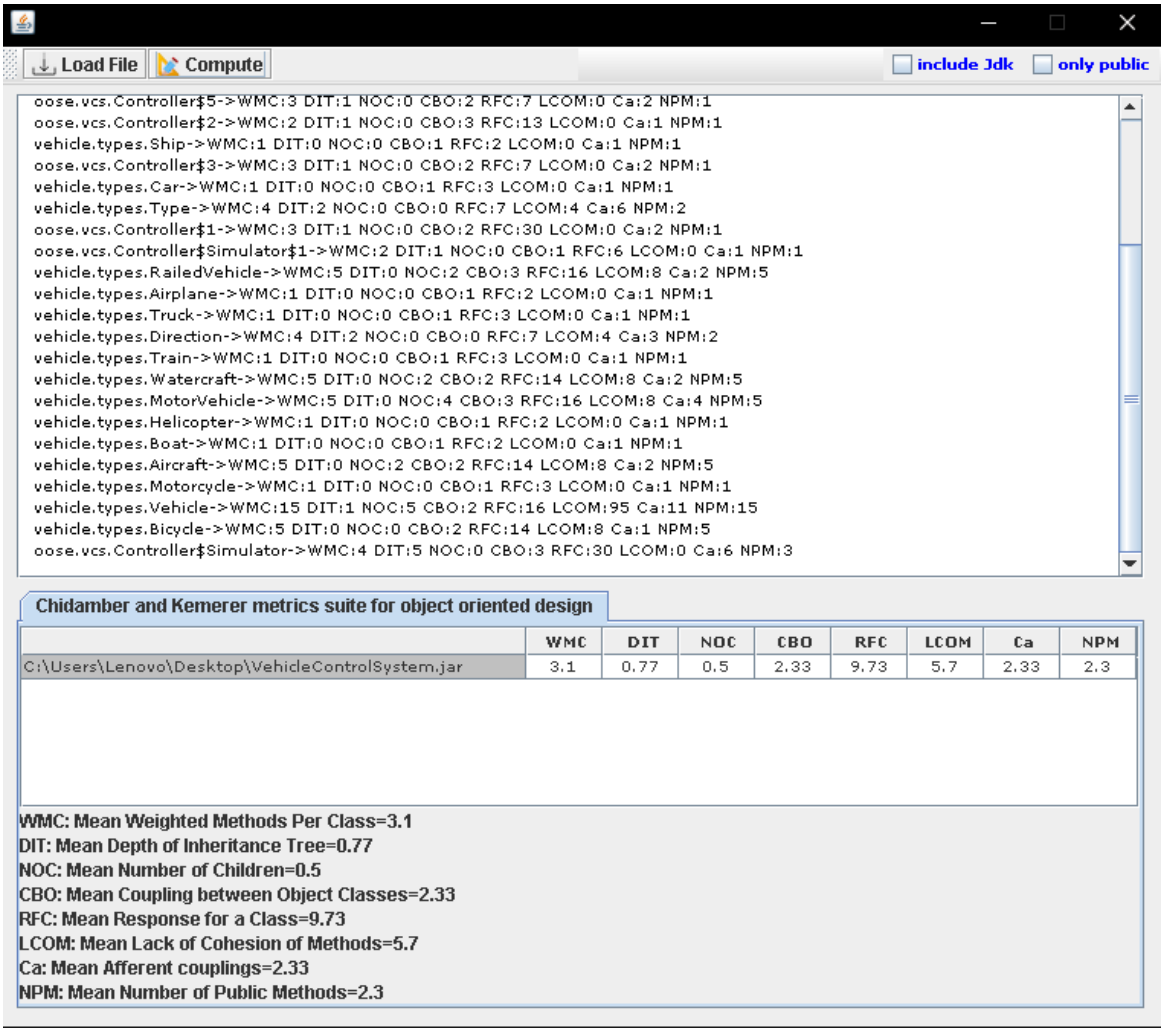
4. (Note: To help in explaining, a screenshot of the original ckmetric values of VCS before refactoring is firstly provided below. As I implement each refactoring, I'll display an image of the new ckmetric values, factoring in only that specific change. In the end, I'll provide the final change in the value of ckmetric, taking into account the combined effect of all 4 refactorings.)



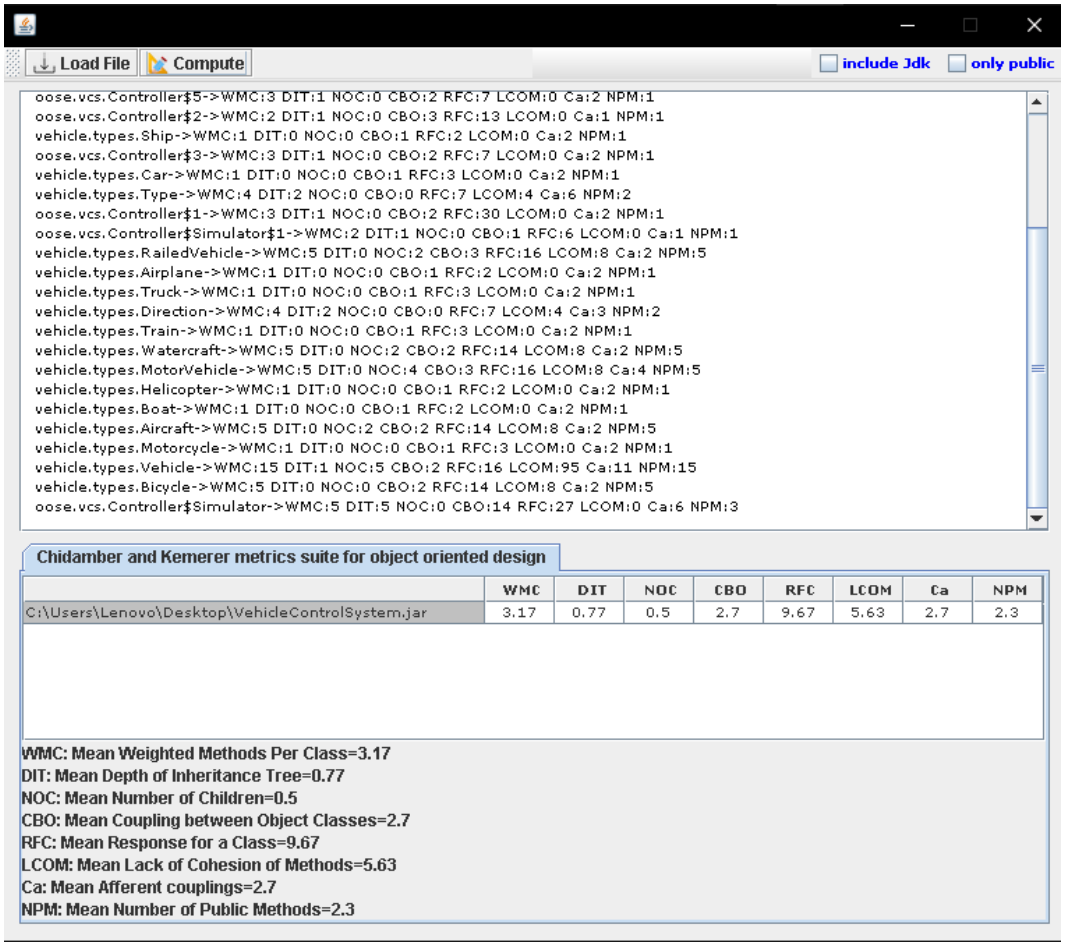
The simulator that hooks to the controller currently has a getPreferredSize() method. In inspection of the code, the presence of this method has little to no effect on the overall functionality of how the VCS operates. As such, for the first refactoring, I have chosen to omit it. In realising that the method simply increases the overall number of methods without much significant benefit to the code, removing it will lead to a reduction in WMC, thus reducing complexity (as shown below).



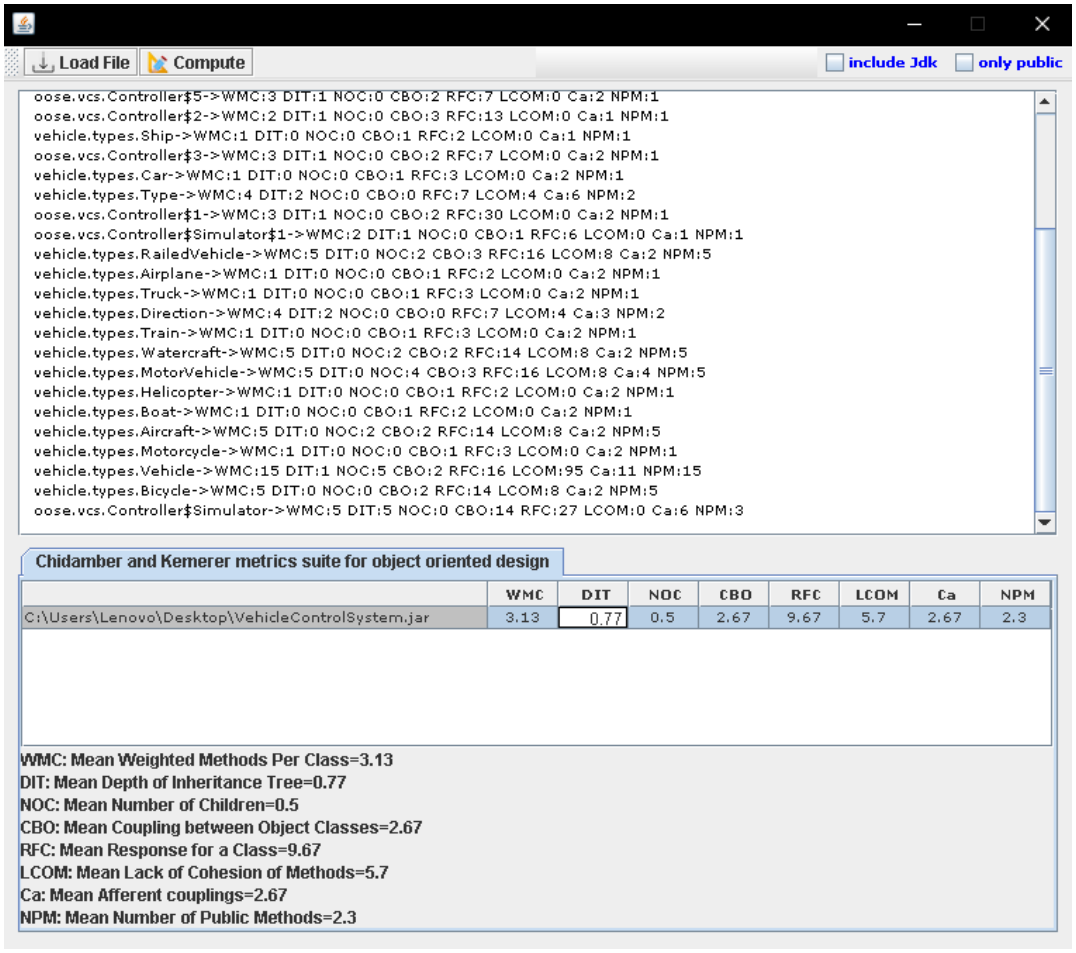
The second refactoring constitutes modifying the existing setDisplayObject() method which implements an overly long if-else chain to open the vehicle image files. This operation can be simplified by removing the method completely and replacing it with a single line of code (commented further on in the refactored implementation). In doing so, complexity should reduce which is shown as a decrease in WMC in the figure below. Additionally, modularity and encapsulation have improved, represented as a drop in CBO value.



For the third refactoring, I created a method to reduce the number of repeats of the code block for changing buttons colours. While this did result in a slight increase in WMC (i.e. a little increase in complexity), the more significant decrease in LCOM justifies this as modularity and encapsulation have improved (and complexity overall decreased).

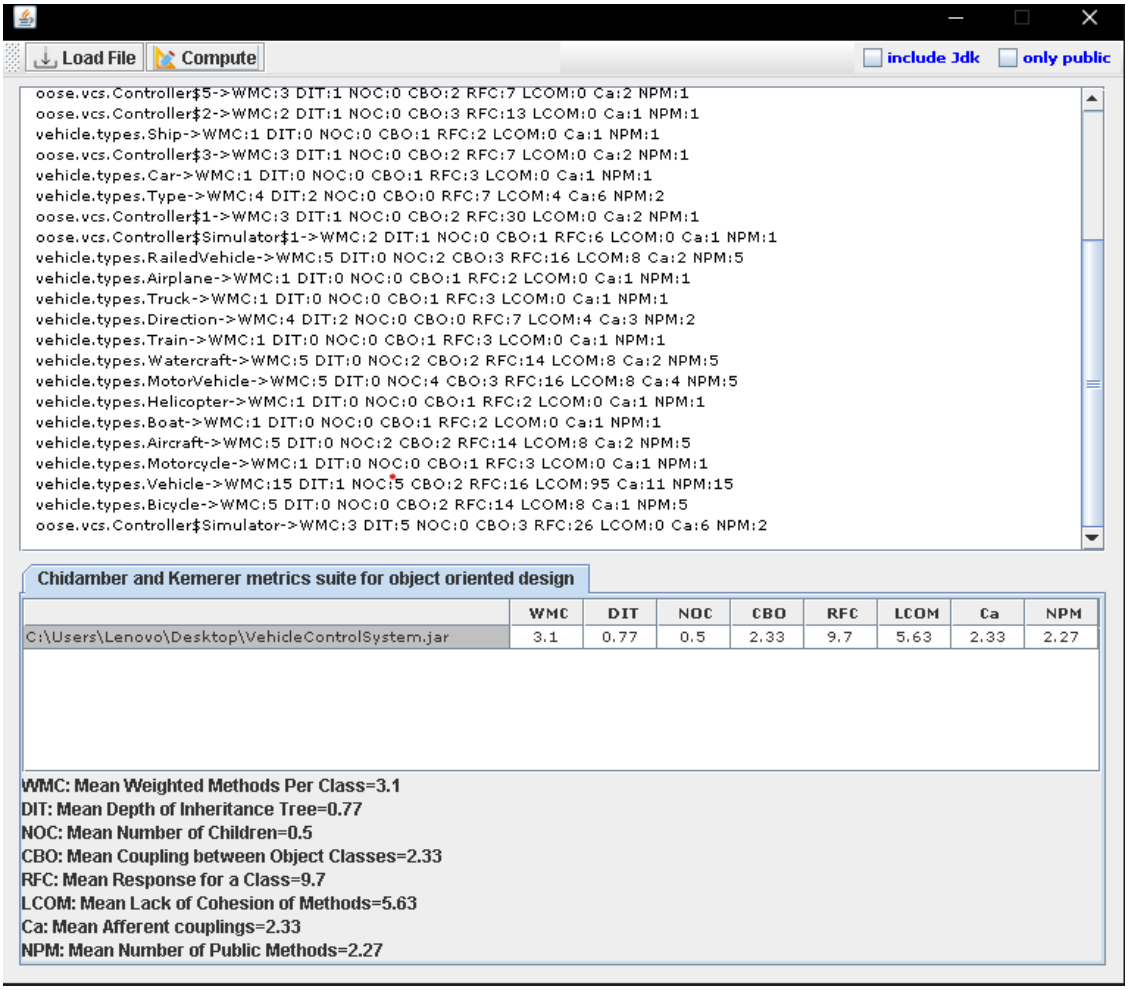


In the fourth refactoring, I substituted the long if-else chain in the method initialiseVehicle(..) with a HashMap. Consequently, CBO value decreased thus attaining reduced complexity while promoting encapsulation. By removing the need for if-else conditions, having less coupled objects makes any future vehicle installments later on simpler to instantiate.



Having justified in the previous paragraphs, here is the final result for ckmetrics:

- WMC reduced (-0.03) - CBO reduced (-0.37) -LCOM reduced (-0.07)



5. The main differences in the modified UML diagram can be noted in the absence of the former two methods getPreferredSize() and setDisplayObject(), and the inclusion of a new method changeButtonBGC(..). The vehicle class and its subclasses have been left as they were since the major class to be refactored has been Controller. To provide more information, the method changeButtonBGC(..) takes in a single parameter which is the Jbutton; its return value is void as its main purpose afterall isn't to return anything but to ensure that the colour of the buttons in the GUI menu are what would be expected (the active vehicle action is highlight in a green background and the rest are in light-gray).

(For more information about the code refactorings and to help navigate to see where refactorings occurred in the code, comments are provided for convenience.)