1)

Graphical user interface, application

Description automatically generated with medium confidence

A picture containing accessory

Description automatically generated

2)

1. To decrease **complexity,** we have to decrease the Weighted Methods per class (**WMC)** which is the sum of the complexities of methods in the class.

* These complexities for each method in the classes a cyclomatic complexity according to McCabe’s Cyclomatic metric

Text, letter

Description automatically generated

* To do this, the number of Edges should be decreased as much as possible

1. To promote modularity and encapsulation, we have to reduce coupling between object classes (**CBO**) — a **class A** is coupled to another **class B** if A affects B or B affects. This means that the extent to which classes affect each other has to be reduced
2. Achieving low Lack of Cohesion (**LCOM**), which means having high cohesion, will decrease complexity and promote encapsulation. Lack of Cohesion can be reduced by decreasing the difference between the number of methods that have no instance variables in common and the number of methods that do.

3)

After ckmetrics analysis, the most important class to refactor is Vehicle because as it has highest values of **LCOM**, **WMC**, and **CBO**. Vehicle.class appears to implement methods and functions not used by all of its subclasses, such as Vehicles#drive which is not used by the Bicycle.class, and Vehicles#street which is not used by the Railed.class and Motor.class vehicles.

Vehicles has a lot of functionality which could be divided into further classes, it also invokes methods in the constructor of each of its subclasses such as Vehicles#setName which could rather be replaced by a Vehicle constructor and the use of super().

Another class which has high ckmetric values is Controller, it has high coupling through the implementation of a simulator class which depends on the instance variables of Controller, it also consists of a lot of repetition in terms of running the kinematics of the vehicle.

4) The suggested refactoring’s to be implemented:

1. Dividing **Vehicle.class** into further subclasses:
   1. **MovableObject**: generalization of an object that moves
   2. **Vehicle**: Vehicle is a Movable object with name and type
   3. **DrivableVehicle**: Vehicle which drive and have objects such as wheels or carriages associated
   4. **CraftVehicle**: vehicle which is a craft that uses steering
2. Removing the use of setters in the constructors of subclasses, and replacing that with initial constructors in the abstract Vehicle classes, which rely on the use of super() or direct access of the instance variables
3. Reducing repetition of similar overridden methods in vehicles, by implementing them in the abstract classes and adding further instance elements to allow that if necessary such as adding a speed unit variable rather than overriding printCurrentSpeed numerous times
4. Deleting unused setters which are replaced by direct initialization of instance elements in the constructors, as well as setters which could disrupt the programs constraints as in **setStopped()** wouldn’t change the velocity, but the method **stop()** would.

Encapsulation means hiding the data in the class and only allowing access through the use of methods, this was done by making sure all the values in Vehicle and any of its super or subclasses are either private or protected. Some setter methods were also removed as they were being used in other methods that manage the values in the class such as setStopped and stop. This also reduces the complexity of the program as the **WMC** of Vehicle decreased from 15 to 4 and was very low for all its super and subclasses.

Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules, by dividing the vehicles into further subclasses associated with the specific functionality, such as implementing DrivableVehicle.class and CraftVehicle.class, as well as MovableObject, each class is responsible for specific tasks, this increases usability and decreases complexity since MovableObject, for example, is a generalization of any object which needs to move. This is noticeable as the **LCOM** in Vehicle decreased from 95 to 0, and became 0 for all of its subclasses thus highly decreasing complexity

After refactoring also, the Simulator and controller classes, the total ckmetric analysis showed a drop in **LCOM** from 5.7 to 0.7, a drop in **WMC** from 3.13 to 2.48, and lower **CBO** as it decreased from 2.7 to 1.85

5)

Diagram

Description automatically generated

The noticeable structural differences appear where *Vehicle.class* has a new superclass *MovableObject.class*. This superclass is used to manage moving objects in general, and the Direction is now associated with the MovableObject.

*Vehicle.class* also has 2 new subclasses *CraftVehicle.class* and *DrivableVehicle.class*, where the Watercraft, Aircraft, and Bicycle are now subclasses of CraftVehicle; whereas MotorVehicle and RailedVehicle now share a field noObjects and are subclasses of DrivableVehicle.

It is quite noticeable that each class looks like it has less instance variables and that the classes are better categorized.

Another difference is the *Controller.class* having a *ControllerUI.class* superclass which manages all the user interface, Controller also has much less instance variables which were replaced by acceleration as it can determine whether vehicle is cruising or accelerating or decelerating by being positive, zero, or negative. The simulator class was separated from the controller class, and is now only connected once to the controller class, thus reducing coupling. This was done by adding a vehicle variable in the constructor and modifying setDisplayObject to take a vehicle as an input.