Worksheet 7: Object Orientation & Aggregation

Updated: 25th April, 2016

The objectives of this practical are:

- Implement the basic object orientation concepts.
- Understand how a class is dependant on its class fields.
- Understand Constructors, Accessors, and Mutators.

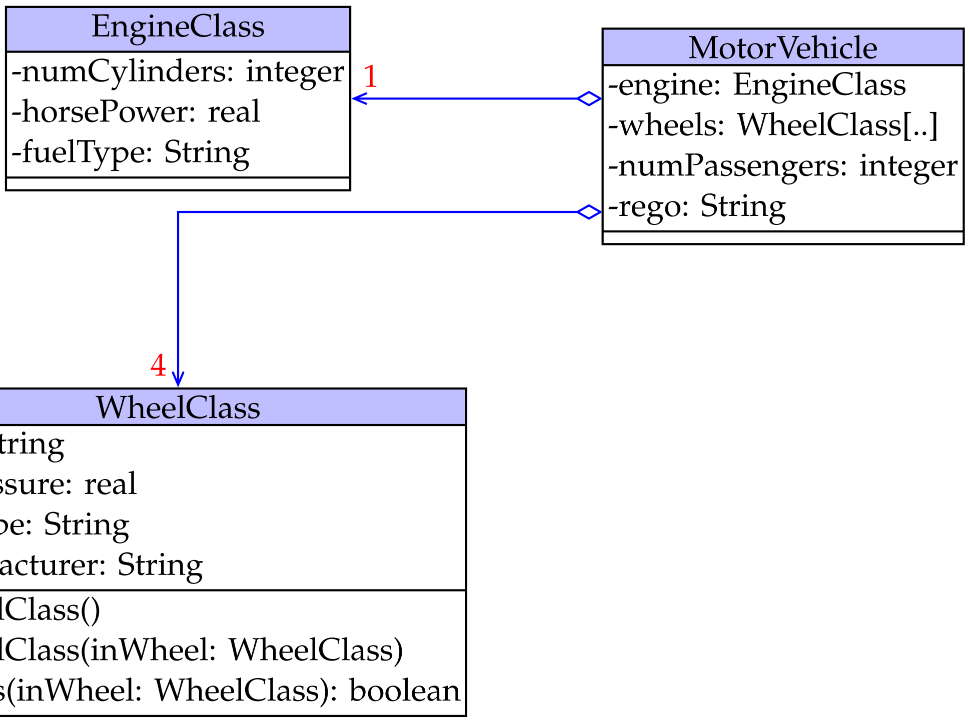
1. Background

Over the next several weeks you are going to be developing (**and refining**) a series of classes used to model motor vehicles. You will need to use the ideas and concepts presented in the **lecture notes, textbook, and examples** in order to complete these practicals.

Note: WheelClass has been provided for you as a complete worked example, before attending your practical you should have spent some time familiarising yourself with the pseudocode design, the java implementation and the provided test harness.

You could even create your own test harness and experiment, after all, you learn the most by causing errors.

The goal of this weeks practical is to develop EngineClass and MotorVehicle. The following diagram shows how these classes interact and provides a brief synopsis of each class.



2. Talking Object Orientation

Before we start developing any of these classes lets talk about Object Orientation.

Form a group of 4 with the people around you and together spend 20-30 minutes discussing these questions. The idea of this exercise is to get you actively thinking about the problems you are about to face. The answers to these questions will also help you design your EngineClass class.

Warning: You need to completely read this worksheet before attending the practical and starting this activity.

Each question also has several sub points that your discussion should address.

- What is the difference between a class and an object?
- What is the purpose of a constructor?
  - How are they related to class fields?
- Why do we need accessors?
  - What relationship do accessors have with class fields?
  - What class fields would you need for EngineClass?
  - How is an accessor different if the class field is an object?
- Why do you always write an equals method?
  - How many equals methods should you write?
  - What fields should be compared in an equals method?
  - Is it appropriate for an equals method to FAIL?
  - How many equals methods do you need for EngineClass?
- Why do we need mutators?
  - How are mutators related to class fields?
  - Why do you **not** need a mutator that changes all class fields at once?
- Why might we use submodules for validation?
- Why are class constants declared as public where as declaring class fields as public is very bad (instant zero)?
  - Where and why did WheelClass use constants?
  - Does engine class require any constants?
- Why should container classes not have INPUT and OUTPUT?

3. EngineClass

The first class you should develop is EngineClass, as shown in the above diagram. You will need the following class fields.

Field	Type	Valid Values	Notes
numCylinders	integer	>=4 multiple of 2	
horsePower	real	>0.0 <=640.0	
fuelType	String	"ULP" "PULP" "DIESEL" "LPG" "98RON"	

Using pen and paper, develop boolean expressions to validate each of these fields (the boolean expression only, no control structures). If your booleans are simplified by constants you should make a note of them now. Once you are comfortable with the booleans turn each into a validation submodule. **You should have these submodules checked by a tutor.**

You should also develop the boolean expression to determine equality (equals method), two EngineClass objects are considered equal if and only if they have the same number of cylinders, horsepower and fuel type.

Now that you understand more about the class field you can begin your **pseudocode** design. Begin by specifying the class name, class fields and any constants, use WheelClass as a format guide.

Next develop the default constructor, remember the default constructor should take no parameters and construct a valid object, what constitutes appropriate values for a default constructor is left to your discretion.

Once the default constructor is complete we will skip to the copy constructor. A copy constructor should take in an object of the same class, use the accessors to retrieve the object state and initialise the class fields (see the lecture notes).

Now complete your alternate constructor, remember to deal with validation as necessary and to indicate when/if the submodule can *FAIL*.

Complete your class by writing accessors and mutators as appropriate. Don't forget the equals and toString methods.

Once your design is **checked by a tutor** you can convert your design to java.

4. Testing your EngineClass

Using the lecture notes and WheelClassTestHarness as a guide, you need to develop a test harness for your EngineClass. Your test harness should test (either directly or indirectly) every submodule you wrote for EngineClass.

5. MotorVehicle

The next class you need to write is MotorVehicle, to do so you will need to implement aggregation, as explained in the lecture notes and textbook.

A MotorVehicle needs an engine (EngineClass), a set of wheels (WheelClass), and a registration/number plate (a string that cannot be validated), you also need to keep track of how many passengers (excluding the driver) the vehicle can hold.

Note: Be mindful that different types of motor vehicles have a different number of wheels.

Two motor vehicles are equal if they have the same engine, the same set of wheels, and the same number of passengers.

Given this information you need to develop constructors, accessors, and mutators as appropriate. It is a good idea to sketch out your field definitions before you start on any algorithms. You should check your design with a tutor before you begin java implementation.

Note: Remember, to maintain the encapsulation of class fields your container classes should be returning *copies* of objects. Not the objects themselves.

To avoid ArrayIndexOutOfBoundsException, your equals method should first check the number of wheels, if they are the same you can then compare each wheel without worry about the array index.

Again you need develop a test harness for MotorVehicle.

Warning: Next week we will be adding inheritance to this class design so you **must** have both EngineClass and MotorVehicle completed before then.

End of Worksheet