Vehicle Java Exercise

You are to write a program to track vehicles and their parts.

A vehicle is composed of an engine. A vehicle has a manufacturer, a model number, a year and VIN number.
An Engine has a size (example: 1000, 250, 750 are typical sizes) and unit of measure (cubic centimeter or cubic inch) and a set of pistons. When the engine is created (using its constructor) the number of pistons is specified. An engine's number of pistons is fixed.
Each piston has a diameter and a condition (excellent, good, poor).

Your vehicle has the following methods:
Get/set manufacturer, model number, VIN and year.
AddEngine, removeEngine and getEngine.

The engine has get/set for methods for its size.
The engine has methods getting and setting a piston's condition. It takes the piston's number and condition

The Piston has a method for getting and setting its diameter.

Part 1
Your main program should:
1. Create a new vehicle of a certain manufacturer, model, VIN and year.
2. Create an engine of a certain size
3. Create a set of pistons for that engine. The engine should encapsulate the Pistons, so the `addPiston()` method only takes the piston number, its condition and its diameter. The engine creates the Piston internally. The main program never actually sees a piston object.
4. Your program should be able to print out the specifications on a vehicle:
   Manufacturer, Model, Year, Engine size, Engine unit of measure, Number of pistons, Piston Diameter and Condition of each Piston.

Part 2:
1. Once you have the above working, modify the engine to calculate the size based on the diameters of the pistons. You should add up the piston diameters and multiply by 10 to get the cubic centimeter size and multiple by 15 to get the cubic inches size.
2. Rerun your program – it should still work correctly.

Part 3 – Use of a Manager Class and Collections
1. Once you have all the above working, create a VehicleManager class to hold multiple vehicles.
2. You should code methods to add, remove, first, next, previous, last methods.
3. Allow the Vehicle Manager to search for a vehicle based on VIN number or by model number.
4. If you search on VIN, only return one vehicle. Print out the specs for this vehicle.
5. If you search by model number, return an array of the objects that you found. Print out the specs for each Vehicle.

Part 4 - Inheritance:

1. Create one subclass of vehicle called motorcycle and one called car.
2. Add attributes and methods to vehicle to track: number of wheels, number of seats, top speed.
3. Add attributes and methods to motorcycle to track type of shift (normal or European), number of riders, number of saddlebags and the saddlebag storage capacity.
4. Add attributes to the car class for whether it has a moon roof, number of airbags and size of the trunk.
5. Create a new subclass of Engine called Carburetor Engine. It should have an attribute of number of carburetors.

6. Create a new subclass of Engine called FuelInjectedEngine. It should have an attribute of number of injectors.
7. Create a new attribute on Engine called float flowRate. It should default to 0. It should have a set and get methods.
8. Create a new method on Engine called getHorsePower(). It is calculated by multiplying the number of pistons by the flowRate.
9. In each Engine subclass, override methods in the Engine class to:
    a. set the flowrate based on either the number of carburetors or the number of injectors.
    b. Calculate the flowrate using the following algorithm:
        - fuel injected engines are number of pistons * nbr of Injectors * 1.3
        - carbureted engines are number of pistons * nbr of Carburetors * 0.8

Part 5 - Abstract Classes:

1. Make the Engine class abstract.
2. Make the Vehicle class Abstract.
3. Add a new method to Vehicle called calcStorageSpace(). This method should be abstract.
4. In the Motorcycle class, override the calcStorageSpace method from Vehicle. Calculate space as the sum of the storage of the saddlebags.
5. In the Car class, override the calcStorageSpace method from Vehicle. Calculate space as the size of the trunk.

Part 6 - Interfaces:

1. Create the Interface Horsepowerable.
2. This interface will have one method: getHorsePower().
3. Create a new class called Animal. It has an int numberOfLegs and a float weight.
4. Create a new class called Rocket. It will have a float fuelRate (values are in the range of 300 to 1000) and a float weight (in tons).
5. Have the Engine, Animal and Rocket classes all implement the Horsepowerable Interface. They must all implement the method get HorsePower().
6. For the Animal class, this is calculated as weight / nbrOfLegs * .6
7. For the Rocket class, this is calculated as fuelRate * 10 / weight.
8. In the main, create an Array of HorsePower Objects.
9. Create 5 new objects of either Animal, Rocket or Engine. Store these in the array.
10. Iterate through the array and print the horsepower of each object.
11. Now do the same with an ArrayList.