# Practice Coding Assignment 13 : Vehicles

**This assignment is entirely optional and designed to give you practice writing code and applying lessons and topics for the current module.**

This homework deals with the following topics:

- Inheritance
- Overriding
- Access modifiers
- Unit testing

## The Assignment

In this assignment, you will implement a class called *Vehicle*, which represents a vehicle. You will also implement two classes that extend the Vehicle class: *Car* and *Bike*. These classes represent different kinds of vehicles. In this program, the *Vehicle* class is the superclass (or parent class), and the *Car* and *Bike* classes are the subclasses (or extended classes, or child classes).

A vehicle will have attributes such as type, brand, and age, which the *Car* and *Bike* classes will inherit. A vehicle will also be able to do things like refuel and run, but different kinds of vehicles will do them differently. Thus, methods like “refuel” and “run” will be defined in the Vehicle class, but you’ll have to override them in the *Bike* and *Car* classes. (Remember, to override a method, create a method in a subclass having the same name and the same number, types, and sequence of parameters, as another method in the superclass.)

## Instance Variables

The Vehicle class has the following instance variables defined for you:

- int currYear - The current year 2020
- int age - The age of the vehicle. For example, if a car was bought in 2000, the age would be 20.
- int gasRemained - The amount of gas remaining in the gas tank. Initially, this is set to 0.
- int gasConsumedPerHour - The amount of gas consumed per hour when the vehicle is running. For the bike, this should be set to 0. For the car, it’s set to 10.

- int totalGasConsumed - The total amount of gas consumed by the vehicle. Initially, this is set to 0.
- int maxGasAmountInTank - The maximum amount of gas that the vehicle can hold. For the bike it's set to 0. For the car, it's set to 200.
- String type - The type of the vehicle. This is either "Car" or "Bike".
- String brand - The brand of the vehicle. For a car, this would be set to something like "Benz" or "Jeep", and for a bike, something like "Trek".

Note, all of these instance variables are defined with the keyword ***protected***. This means they are accessible from anywhere within the *Vehicle* class and from within the Car and Bike subclasses.

## Constructors

The constructors in the *Vehicle*, *Car*, and *Bike* classes have all been defined and **implemented for you**. You don't need to make any changes to the constructors in any of the class files.

## Methods

There are 2 methods defined in the *Vehicle* class that **need to be overridden in the *Car* and *Bike* classes**:

- refuel(int amountOfGas) - For a car, this method adds the given amount of gas to the gas tank. For a bike, we don't need to refuel, so this method only prints: "You don't need to refuel a bike."
- run(int hours) - For a car, this method tells the car to run for the given number of hours. For a bike, we don't need to run, so this method only prints: "Eco-friendly travel by bicycle!"

Each method has been defined for you in the *Vehicle* class, but without the code. See the javadoc for each method for instructions on what the method is supposed to do and **how to write the code for the overridden methods in the *Car* and *Bike* classes**. It should be clear enough. In some cases, we have provided hints and example method calls to help you get started.

For example, below is the code for the "refuel" method in the *Vehicle* class. The method just returns with no value. Read the javadoc, which explains what the method is supposed to do.

Then write your code to **override and implement the method in the *Car* and *Bike* classes**. You'll do this for the "run" method as well.

```
/**
 * Should be overridden by subclasses Car and Bike.
 *
 * For the bike, we don't need to refuel.
 * Therefore, when bike calls refuel, nothing should happen. Just
 * print: "You don't need to refuel a bike."
 *
 * For the car, you have to make sure there's room in the gas tank
for
 * the given amount of gas.
 *
 * @param amountOfGas to put in the gas tank
 */
public void refuel(int amountOfGas) {
    return;
}
```

Note, you don't need to implement the "refuel" method in the *Vehicle* class itself, since we're overriding it in both the *Car* and *Bike* classes.

There are another 2 methods defined in the *Vehicle* class that **need to be implemented in the *Vehicle* class**, but **do not need to be overridden in the *Car* and *Bike* classes**:

- equals(Object o) - Compares two vehicles for equality by comparing their brand and their type
- toString() - Returns the type and the brand of the vehicle as a String

Again, each method has been defined for you in the *Vehicle* class, but without the code. See the javadoc for each method for instructions on what the method is supposed to do and how to write the code. It should be clear enough. In some cases, we have provided hints and example method calls to help you get started.

For example, below is the code for the "equals" method in the *Vehicle* class. The method compares two vehicles for equality. Read the javadoc, which explains what the method is supposed to do. Then write your code to **implement the method in the Vehicle class itself**. You'll do this for the "toString" method as well.

```
/**
 * Compare two vehicles are equal by comparing their brand and their
 * type.
 *
 * @return whether the two vehicles are equal
 */
@Override
public boolean equals(Object o) {
    // TODO Implement method
    return false;
}
```

There are other methods defined in the *Vehicle* class that have **already been implemented for you**:

- getAge()- Gets the age of the vehicle
- getGasRemained() - Gets the gas remaining in the tank
- getTotalGasConsumed() - Gets the total gas consumed by running

You don't need to implement these simple "getter" methods. (Remember, in Java, "getters" and "setters" are conventional methods that are used for retrieving and updating the value of a variable. Typically, getters and setters provide access to private or protected variables.)

## Unit Testing

In addition, you will write unit tests to test all method implementations in the program. Each unit test method has been defined for you, including some test cases. First make sure you pass all of the provided tests, then write **additional and distinct test cases** for each unit test method.

For example, we have defined a "testRefuel" method for you (see below) which tests the "refuel" method. Pass the tests provided then write additional tests where it says "// TODO". You'll do this for each unit test method noted in the testing file.

```
@Test
void testRefuel() {
    bike1.refuel(100);
    assertEquals(0, bike1.getGasRemained());
```

```
    jeep1.refuel(100);
    assertEquals(100, jeep1.getGasRemained());
    jeep1.refuel(101);
    assertEquals(200, jeep1.getGasRemained());

    // TODO write at least 3 additional test cases

}
```

## Main Method

The main method calls have been provided for you (see example below). They are designed purely to call the other methods in the program, and to help you run the program. Nothing needs to be completed in the main method.

```
public static void main(String args[]) {

    //create a car that is a jeep from 2000
    Car jeep = new Car("Jeep", 2000);
    System.out.println("jeep age : #" + jeep.getAge());

    //call methods on jeep
    jeep.refuel(1000);
    jeep.run(10);
    jeep.refuel(100);
    jeep.run(1000);
}
```

## Tips for this Assignment

In this assignment, some tips are given as follows:

- String equality:
    - To compare Strings, use the *equals()* method
    - Recall "==" is used for comparing primitive values, but Strings are Objects, and Object equality is tested with the *equals()* method
    - For example:

```
String strOne = "Hello";
String strTwo = "hello";
```

```
//stringsAreEqual would be false
//because strOne is capitalized and strTwo is not
boolean stringsAreEqual = strOne.equals(strTwo);
```

## Submission

You have been provided with *Vehicle.java*, *Car.java*, *Bike.java*, and *VehicleTest.java*. To complete the assignment, implement the methods in *Vehicle.java*, *Car.java*, and *Bike.java*, making sure you pass all the tests in *VehicleTest.java*. Then write **at least 3 additional and distinct test cases** for each unit test method noted in *VehicleTest.java*. Do not modify the name of the methods in any of the files or the automated testing will not recognize it.

You will submit four files for this assignment: *Vehicle.java*, *Car.java*, *Bike.java*, and *VehicleTest.java*. Make sure your program and the unit testing files run without errors! Submit the completed program using the steps outlined in the assignment in Coursera.

## Evaluation

**This assignment is evaluated, so that you can assess how well you understand the material. However, the evaluation of the assignment will not affect your course grade.**

*Points:*

1. Does your code function correctly? (16 pts)
    - refuel(int amountOfGas) - 4 pts
    - run(int hours) - 4 pts
    - equals(Object o) - 4 pts
    - toString() - 4 pts
2. Did you include at least 3 additional distinct and valid test cases for each test method? Do all of your tests pass? (12 pts)
    - testRefuel() - 3 pts
    - testRun() - 3 pts
    - testEquals() - 3 pts
    - testToString() - 3 pts