ICS 141

Lab #7

Problem Description:

In this lab you will be designing an abstract class and extending it to a specific class. The abstract class you will design will be called the Vehicle class. The concrete class that you will design is called the Car class.

As mentioned, the Vehicle class must be defined as abstract. Recall that this means we cannot make an instance of the Vehicle class. This makes sense as we do not necessarily want to represent a generic ‘vehicle’, but rather, we would instead represent actual vehicles, such as planes, trains, and motorcycles.

What do we need to represent in the Vehicle class? All vehicles will have a color. Any vehicle would have a year of manufacture. All of these general items are defined below.

A variable to store the exterior color of the vehicle.

A variable to store the name of the vehicle. I.E. “Dodge Neon” or “Harley Davidson” would be examples.

A variable to store the year the vehicle was manufactured.

An abstract print method for describing the vehicle’s color

An abstract print method to describe the vehicle in general (name, color, date of manufacture).

A constructor which sets the above-mentioned private data to passed parameter values. Therefore, your constructor should take 3 parameters, corresponding to the private data of the Vehicle class.

Now, the Car class, which must extend this abstract Vehicle class, will have the following properties:

A variable to store the maximum MPH (miles-per-hour).

A variable to store the interior color of the car.

Implement the abstract method for describing the car’s colors. Use a simple void method and print to the console. Be sure to state the color of the interior and exterior.

Implement the abstract method for describing the car. Use a simple void method and print to the console. For a car, be sure to state the maximum MPH, the year the car was manufactured, and the name of the car.

A constructor which sets all of the private data from the Vehicle class as well as the private data from the Car class. Set these using passed parameter arguments. Therefore, your constructor should take 5 parameters, corresponding to the data fields of the Vehicle class as well as the data fields of the Car class.

Hints:

Given the description, you should have at least two classes defined. Of course, you can also have an additional class for testing purposes if you choose.

Remember that you do not implement abstract methods in the abstract class itself. You simply define them so that they can later be implemented by the subclasses of the abstract class.

The constructor for the abstract class should be declared with the **protected** modifier. This way, only the sub-classes can access the constructor (As well as other classes in the same package).

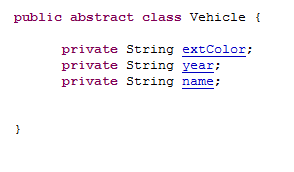
In the subclasses, we need access to the private data of the abstract class. What is the best way to do this? Getters/Setters in the abstract class? Should we make the data protected instead of private?

**Deliverables:**

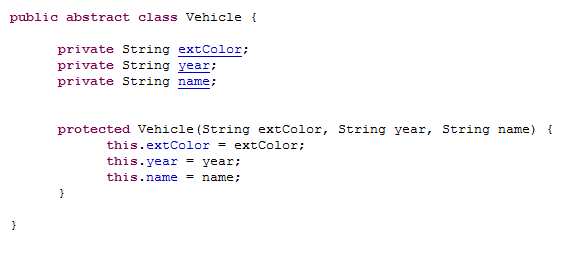
The .java file(s) for this lab. These are the files that contain your source code. I do not need the .class files. Submit the java file(s) to the lab #4 drop box by Monday, April 6th.

**Walkthrough**

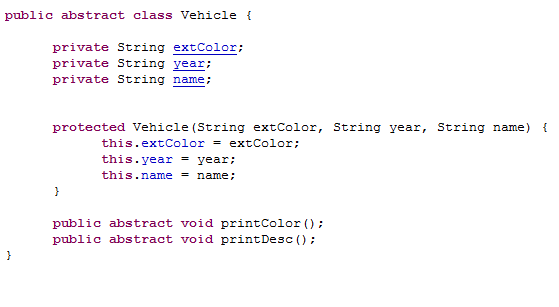
Because the Car class is dependent on the Vehicle class, it is logical that we should design our Vehicle class first. Given the above description, we know that we’ll have three data fields: The exterior color, the name, and the year of the vehicle. We’ll set these to private so that they are properly encapsulated. We’ll also go ahead and declare our class to be abstract using the abstract keyword. Remember, this means we’ll not be able to make instances of the Vehicle class!



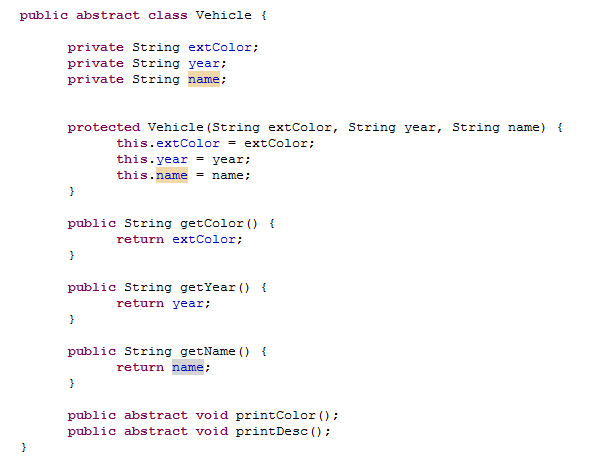
Now we’ll define our superclass constructor. We still need to provide a constructor so that it can be used by the subclass in its constructor. Note we use the protected modifier. This way, only the sub-classes can access the constructor (As well as other classes in the same package).



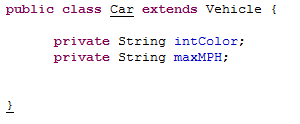
The next step is then to declare those two abstract methods. Remember, we only declare these, we do not implement them in the superclass.



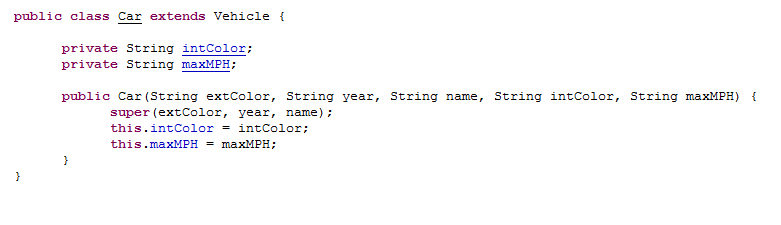
The last step for this class is to define a few getters, so that we’ll have access to these fields within the subclass. We’ll define all of these methods as public so that they’ll be accessible everywhere.



The next step will be to design our Car class. We’ll have this class inherit directly from the Vehicle class as it has much of the groundwork laid out for us. We’ll also go ahead and define our two extra data fields.



The next step will be to define our constructor. We’ll make it public this time as we may need access to this constructor from anywhere in Java. We’ll call the superclass constructor to set those three fields and also set our two fields accordingly.



Almost done. Now we need to fix that last error. Here, if you mouse over Car, you’ll see the compiler is complaining that you have not implemented those two abstract methods. Therefore, this is the final step. Note the syntax for defining our two previously abstract methods. Now that we are providing the actual implementation, we omit the abstract keyword (we are declaring the two methods as concrete at this point). Other than the abstract keyword, the method header must be the same. Again, the main method here is for easy testing. Experiment with these two classes a bit. See how changing the modifiers affects the whole program.

