Introduction to Java
CS9053 Section I
Thursday 6:00 PM – 8:30 PM
Prof. Dean Christakos
Feb 17th, 2023
Due: Feb. 24th, 2023

Assignment 4

Part I: Inheritance

1. Here is a set of entities:

   Vehicle
   Motorcycle
   Car
   Bicycle
   CargoCycle

   I'm not going to write out full UMLs and hierarchies because your assignment is to figure out how to implement it yourself.

   A Vehicle is any vehicle with wheels and has a color and has a certain amount of cargo space. It has attributes like number of wheels and cargo space (in cubic feet or liters. Units don't matter). Cars have 4 wheels and cargo space. Cars also have either 2 or 4 doors. Motorcycles have an array of strings that list accessories (things like "helmet clip", "grip warmers", "usb charger", etc. There are no fixed values for this). Bicycles and motorcycles have 2 wheels and 0 cargo space. Cargocycles are a type of bicycle and can have 2, 3, or 4 wheels and have cargo space. Bicycles and CargoCycles can be electric.

   Cars have a method called "PressGasPedal" which returns a string called "accelerating". Motorcycles have a method called "TwistThrottle" which does the same thing. Bicycles and CargoCycles have a method called "Pedal" which returns a string "pedaling."

   Your first step is to implement this hierarchy. In constructors with args, you should use `super()` constructor with args to set the values in superclasses. There should be no redundancy, or at least as little as possible

   `toString()` should contain the name of the class and all of the fields/data for that class. You can decide whether this will work by a subclass calling the `toString` method of the superclass and including that in the subclass `toString` result (as in the

`GeometricObject` hierarchy) or if the `toString` method accesses all the data/fields in the object itself.

This uses the static "id" pattern. Every creation of a new Vehicle object (of any kind) should generate a new and unique id from the static `next_id` field, which is stored in the `id` field and accessible by `getId`

2. Now that you've implemented this, write `equals` methods for all the classes. The equals method should take an `Object` as an argument and return `true` if the field values of the class and its superclass(es) are equal.

Part II – Arraylists

1. Create an `Arraylist` of `Vehicle` objects.

    a. Create 8 Vehicle objects:

    1 one red motorcycle with accessories “grip warmers” and “usb charger”
    2 blue cars with 4 doors and 20 cubic feet of cargo space
    1 black bicycle
    2 green cargocycles with 3 wheels and 10 cubic feet of cargo space
    1 gray car with 2 doors and 10 cubic feet of cargo space
    1 white car with 4 doors and 25 cubic feet of cargo space

    Put those objects in the `Arraylist`. They should all be able to be added to the same `Arraylist`, regardless of their subclass. The `Arraylist` MUST be parameterized to accept `Vehicle` objects. It should accept all `Vehicle` objects but should not accept non-Vehicle objects. For example, this:

    ```
    vehicleArrayList.add(new Object());
    ```

    should not compile

    b. Print out the average cargo space of all the Car objects in the `Arraylist`. You have to do this by looping through the array list, finding the Car objects, and get their cargo space amount, and then finding the average of their salaries

    c. **Remove the matching objects:** Retain the 1st blue Car object in a Car variable with the blue color, 4 doors, and 20 cubic feet of cargo space. The goal is ultimately to remove all of the Car objects in the `Arraylist` that match this variable:

    To start: Loop through the `Arraylist` and print out (using `toString`) which objects in the Car object is equal (in value) to the Car object in your variable. “Equal” in value means having the same color and doors and wheels and cargo space, but obviously their their Ids do not need to match

    Also print out which object in the `Arraylist` is the **same object** as the one in your variable.

    You must figure out how to remove all the matching objects from the Arraylist. There is no one correct way to do this. But there are incorrect ways.

d. **Print out the remaining objects:** Loop through the Arraylist again and print out (using `toString`) all the remaining objects in the Arraylist.