Introduction to Computer Science II
Assignment 3

Due: 4:00 p.m. June 16, 2016

Extending Classes in Java

Create a system that describes basic transportation system and allows for tracking different types of cars and passengers as follows:

Overview:
A passenger is a person that is described by his/her name and who can be either a driver or a regular person.
If the passenger is a driver, then he must have a valid driving license detailed as follows: class 7 (can only drive 2 wheelers such as bikes), class 5 (can only drive regular 4 wheelers such as regular cars) or class 1 (can drive all types of vehicles including trucks and buses).

A vehicle is any movable entity that can be driven and contains one or more passengers. Any vehicle can be described by its speed, number of passengers (including the maximum allowed), and the number of wheels. For any vehicle to be driven, a qualified passenger driver must be associated with it, and there should be a way to verify if a particular passenger can drive this vehicle or not. The driver can move the vehicle by accelerating/braking as needed. Examples of specific vehicles are powered vehicles (e.g., cars, buses, and trucks) and non-powered vehicles (e.g., bikes).

Powered Vehicles are vehicles that include an engine, and are described using the engine power, and the number of doors. Examples of powered vehicles are cars, trucks, and buses.
Cares are special type of powered vehicles that can additionally be described using their colors and models. Most cars run by a hydraulic engine by default although few of them may contains a diesel engine.
Trucks are special type of cars that runs by diesel and enables transportation of stuff, and are additionally categorized based on their size (e.g., 13.6 square meter).
Buses are special type of powered vehicles that can additionally be described by either having accessibility seats available or not. Buses work with either diesel or hydraulic engines.

Non-powered vehicles simply require manual driving, and are described using their weight. Bikes are special types of non-powered vehicles that can additionally be described using their frame material (e.g., metal).

Create the following classes: Vehicle, PoweredVehicle, NonPoweredVehicle, Bus, Bike, Car, Truck, Passenger, and Test.

The specification for each is as follows:

Passenger: This is a class that extends from the Object class. The following operations on a Passenger object must be supported:

1. One *constructors*: one that allows you to specify the *name* of the passenger. Default constructor should not be allowed.
2. A *pair of get/set* methods that encapsulates the status of this passenger either as a driver or a regular passenger.

3. One *updateLicenseClass* method: sets the license class number of this passenger. Valid values are: -1 (cannot drive anything), or as detailed above.
4. Two *getter* methods: one to return the name of the passenger and the second to return license class he or she is having.
5. A *toString* method: returns a String which shows the current state of the Passenger (typically the name, and if s/he is a driver (including the driving class).

Vehicle: This is an abstract class that extends from the Object class. The following operations on a Vehicle object must be supported:

1. A *set of methods* for keeping track of passengers inside the vehicle (e.g., *addPassenger, removePassenger, getPassengersCount, getMaxPassengersCount, setMaxPassengersCount*).
   Note that the number of passengers should not exceed the max number allowed for this vehicle.
   (Hint: you may use an array for this)
2. Two *setter* methods: one to set the number of wheels and one that sets the maximum number of passengers.
3. One *getDriver* method: searches for and returns the (first) driver among the any existing passengers of the vehicle.
4. Two *getter* methods: one that returns the current speed, and the second return the number of wheels.
5. One *accelerate* method: updates the current speed of the vehicle only when there is at least a passenger with a valid (driving) license, and based on the current speed, the number of wheels and whether an engine exist (abstract; redefined by subclasses).
6. One *brake* method: once applied the vehicle should attempt to reduce speed only when there is at least a passenger with a valid (driving) license, and based on the number of wheels and the current speed. (abstract; redefined by subclasses)
7. One *canBeDriven* method: a method that asserts if this vehicle can be driven by a given passenger (abstract; redefined by subclasses).
8. A *toString* method: returns a String which shows the current state of the vehicle (should be overridden).

PoweredVehicle: This is a concrete abstract class that extends the Vehicle class. Assume that powered vehicles are always driven by powerful engines (whether hydraulic or diesel, with force values ranging between 1600 and 5500). A powered vehicle has a set of values that represent number of doors, and engine power. The following operations on a car object must be supported:

1. Two *setter* methods: one sets the number of doors, and one to set the engine power.
2. Two *getter* methods: one returns the number of doors, and one to return the engine power.
3. A *toString* method: returns a String which represents the current state of the powered vehicle.

NonPoweredVehicle: This is a concrete abstract class that extends the Vehicle class. Assume that non-powered vehicles are always driven manually. A non-powered vehicle can be described by its weight (in kg). The following operations on a car object must be supported:

4. One *setter* methods: one sets the weight.
5. One *getter* methods: one returns the weight.
6. A *toString* method: returns a String which represents the current state of the non-powered vehicle.

Car: This is a concrete class that extends the PoweredVehicle class. Assume that cars are always driven by hydraulic engines (force values range between 1600 and 4500 horsepower). A car can be described using its model and color. The following operations on a car object must be supported:

1. Two *constructors*: that allows you to specify the color, model, number of doors, engine power, and the maximum number of passengers, and one that initializes to red, ford, 4 doors, 2200 horsepower, and 5 people.
2. Three *setter* methods: one sets the color, one sets the number of doors, and one sets the car model.
3. Three *getter* methods: one returns the color, one returns the number of doors, and one return the car model.
4. One *canBeDriven* method: returns true if the passenger's class is 1 or if the class is 5 and the number of wheels less than or equal to 4.
5. One *accelerate* method: increases the speed of the car based on its engine power and the number of wheels. Use the following formula:
   *newSpeed = currentSpeed + engine_power / 1000 * numOfWheels*
6. One *brake* method: reduces the speed of the car based on its engine power and the number of wheels. Use the following formula:
   *newSpeed = currentSpeed - engine_power / 1000 * numOfWheels*
   Note that the new speed should not go below 0.
7. A *toString* method: returns a String which represents the current state of the car.

Truck: This is a concrete class that extends the Car class. Assume that trucks are always driven by diesel engines (force values range between 3500 and 6000 horsepower). Additionally, a truck can be described by its size. The following operations on a car object must be supported:

1. Two *constructors*: that allows you to specify the size, number of doors, engine power, and the maximum number of passengers, and one that initializes to 13 square meter, 4 doors, 5000 horsepower, and 2 people.
2. Two *setter* methods: one sets the size and one sets the number of doors.
3. Two *getter* methods: one returns the size and one returns the number of doors.
4. One *canBeDriven* method: returns true if the passenger's class is 1.
5. One *accelerate* method: increases the speed of the truck based on its engine power and the number of wheels. Use the following formula:
   *newSpeed = currentSpeed + engine_power / 1200 * numOfWheels*
6. One *brake* method: reduces the speed of the truck based on its engine power and the number of wheels. Use the following formula:
   *newSpeed = currentSpeed - engine_power / 1200 * numOfWheels*
   Note that the new speed should not go below 0.
7. A *toString* method: returns a String which represents the current state of the truck.

Bus: This is a concrete class that extends the PoweredVehicle class. Assume that buses can use either hydraulic or diesel engines (force values range between 1600 and 6000 horsepower). A bus can have accessibility seats available or not. The following operations on a bus object must be supported:

1. Two *constructors*: that allows you to specify the availability of accessibility seats, the number of doors, engine power, and the maximum number of passengers, and one that initializes to yes (accessibility seats available), 4 doors, 4500 horsepower, and 2 people.
2. One *setter* methods: one sets the availability of accessibility seats.

3. One *getter* methods: one returns the availability of accessibility seats.
4. One *canBeDriven* method: returns true if the passenger's class is 1.
5. One *accelerate* method: increases the speed of the bus based on its engine power and the number of wheels. Use the following formula:
   *newSpeed = currentSpeed + engine_power / 1100 * numOfWheels*
6. One *brake* method: reduces the speed of the bus based on its engine power and the number of wheels. Use the following formula:
   *newSpeed = currentSpeed - engine_power / 1100 * numOfWheels*
   Note that the new speed should not go below 0.
7. A *toString* method: returns a String which represents the current state of the bus.

Bike: This is a concrete class that extends the NonPoweredVehicle class. A bike can be described by its frame material. The following operations on a bike object must be supported:

1. Two *constructor*: that allows you to specify the weight and frame material, and one that initializes to 20km, and steel.
2. Two *setter* methods: one sets the weight, and the second sets the frame material.
3. Two *getter* methods: one returns the weight, and the second returns the frame material.
4. One *canBeDriven* method: returns true if the passenger's class is any value other than -1.
5. One *accelerate* method: increases the speed of the bike constantly, as follows:
   newSpeed = currentSpeed + 5
6. One *brake* method: reduces the speed of the bike linearly, as follows:
   newSpeed = currentSpeed - 5
   Note that the new speed should not go below 0.
7. A *toString* method: returns a String which represents the current state of the bike.

Test: This is a final class that extends the Object class. It only has a *main*() method, which is used to test all of the above concrete classes. In this client code you **must** do the following:

1. Create two vehicle objects of each vehicle type (e.g., two bikes, two cars, etc.). Initialize each created object with any suitable data of your choice (you do not need to ask user for this data), and note that some of the objects must be created using the default constructor. Also set the maximum number of passengers of the two created cars to 4 and 2 respectively.
2. Store the references to all the vehicle objects in one suitable data structure such as an array.
3. Create 22 passenger objects (with random names).
4. Promote (the first) 7 of the created passengers to drivers as follows: One driver for buses, One driver for trucks, two drivers for bikes, and three drivers for cars.
5. Print simple statistics about the vehicles and passengers you have created so far. (basically print the list of vehicles with their details, as well as the passengers and if they can driver or not along with their driver license whenever possible).
6. Add the bus driver to one of the bus objects, one truck driver to one of the created trucks, one of the bike drivers to the one of the created bikes, and two of the car drivers to the two created cars. Also associate the last car driver to the secondly created bike object.
7. Print the status of the vehicles (including if it can be driven or not).
8. Attempt to add all non-driver passengers to all vehicle objects starting by filling the car objects first and then the other vehicle types. (print the status of a vehicle as full if attempting to add passengers and failed).
9. Move all car vehicles 10 times and all non-car vehicles 8 times, by calling the accelerate method.

10. Attempt to stop all car vehicles 5 times and all non-car vehicles 4 times, by calling brake method on each of them.
11. Compute and display the fastest and slowest vehicle.
12. Delete all vehicle objects that cannot be driven.
13. Print the complete system status: all vehicle objects and all passenger (and drivers) objects.

When using arrays, you may declare them with bigger size than the number of objects you actually need.

Bonus

Use one of the Collections classes to store the vehicles and passenger objects, instead of an array. Possible choices are the ArrayList or HashSet classes. (4 points)

Documentation

Create a UML class diagram that documents all of the above classes (Except Test) and their relationships. Hand-drawn notation is acceptable, although you can use a graphics program if you wish.

Use *javadoc* to create HTML documentation for all classes (Except Test). Be sure all the classes and all the methods are documented; they must be declared public for *javadoc* to work.

New Skills Needed for this Assignment:

- Understanding and use of inheritance, including redefinition of methods and constructors, polymorphism, and abstract and final classes
- Use of the extends keyword
- Appropriate use of the abstract and final keywords
- Understanding and use of aggregation, composition, and association relationships
- Use of the Java arrays to store object references
- Creation of class diagrams using UML (for classes and their relationships)

Submit the following:

1. Your Java source code via electronic submission. Use the *Assignment 3* Dropbox Folder in D2L to submit electronically. Your TA will compile and run your program to test it. Your source code file names must use the names specified above (i.e. the TA will type *java Test* to execute your code).
2. A script showing the compilation of your source code and a sample run through your program. Name this file *script.txt* and also submit it electronically to the D2L drop box.
3. Your HTML documentation produced by running *javadoc* on your source code. Also submit this to the D2L drop box. Your TA will check the HTML documentation to make sure it is complete and correct.
4. Your class diagram using UML notation. Submit this electronically to the D2L drop box. The file should be in a commonly used file format such as PDF, TIFF, or JPEG.

Introduction to Computer Science II
Assignment 3 Grading

Student: ____

Passenger Class

Instance variables	2	______
Passenger Driver (get/set)	2	______
Constructor	1	______
Update method	1	______
Getter methods	2	______
toString method	1	______

Vehicle Class

Instance variables	4	______
Passenger Tracking methods	5	______
2 Setter methods	2	______
2 Getter methods	2	______
Accelerate method	1	______
GetDriver method	1	______
Brake method	1	______
CanBeDriven method	1	______
toString method	1	______

Powered Vehicle Class

Instance variables	3	______
2 Setter methods	2	______
2 Getter methods	2	______
toString method	1	______

Non-Powered Vehicle Class

Instance variables	2	______
1 Setter methods	1	______
1 Getter methods	1	______
toString method	1	______

Car Class

Instance variables	2	______
2 constructors	2	______
3 setter methods	3	______
3 getter methods	3	______
Accelerate method	1	______
Brake method	1	______
CanBeDriven method	1	______
toString method	1	______

Truck Class

Instance variables	1	______
2 constructors	2	______
2 setter methods	2	______
2 getter methods	2	______

Accelerate method	1	______	
Brake method	1	______	
CanBeDriven method	1	______	
toString method	1	______	
Bus Class			
Instance variables	1	______	
2 constructors	2	______	
1 setter methods	1	______	
1 getter methods	1	______	
Accelerate method	1	______	
Brake method	1	______	
CanBeDriven method	1	______	
toString method	1	______	
Bike Class			
Instance variables	1	______	
2 constructors	2	______	
2 setter methods	2	______	
2 getter methods	2	______	
Accelerate method	1	______	
Brake method	1	______	
CanBeDriven method	1	______	
toString method	1	______	
Test Class			
Complete testing in main()	13	______	
Miscellaneous			
Code structure (documentation, formatting, etc.)	4	______	
Elimination of redundant code	3	______	
Class Diagram	6	______	
HTML Documentation	5	______	
Total	**116**	______	______%
Bonus			
Use of one of the Collections Classes	4	______	
Total Bonus (10% for 4/4)	**4**	______	______%
Assignment Grade	**120**	______	______%