CSE 114 Fall 2023: Assignment 9

Due: November 26, 2023 at 11:59 PM [KST]

Read This Right Away

Directions

- At the top of every file you submit, include the following information in a comment
    - Your first and last name
    - Your Stony Brook email address
- Please read carefully and follow the directions exactly for each problem.
- Your files, Java classes, and Java methods must be named as stated in the directions (including capitalization). Work that does not meet the specifications will not be graded.
- Your source code is expected to compile and run without errors. Source code that does not compile will not be graded.
    - You can get partial credit if your program is incomplete but does compile and run.
- You should create your program using a text editor (e.g. emacs, vim).
- You must use the command-line interface to run your programs. That is, you must use the `javac` and `java` commands to do this assignment. Do not use IntelliJ IDEA yet.
- Your programs should be formatted in a way that is readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what a readable style is, see the examples from class and textbook as a starting point for a reasonable coding style.

What Java Features to Use

For this assignment, you are *not* allowed to use more advanced features in Java than what we have studied in class so far.

Problem 1 (10 points)

Develop a class called **Car**.

In Problems 1 through 3, you will be designing several classes. First we will design a class named **Car**; then **PassengerCar** and **SportsCar** each as a subclass of **Car**. Conceivably you could design many more classes as subclasses to model other kinds of cars such as SUV's (e.g., Chevy Tahoe), 18-wheelers (e.g., Kenworth T2000), etc., but we will not worry about them in this problem set.

In Problem 1, we will first design a class named **Car** in a file named **Car.java**. This class is to include attributes that are common in all these different kinds of cars. From the descriptions of each member, you should be able to figure out the correct primitive type! You will include the following attributes (fields) in the **Car** class:

- The **make** of a car such as "Chevy", "Lamborghini", "Kia", etc.
- **model** such as "Outback", "Diablo", "Optima", "280Z", etc.
- **year** such as 2019, 2007, etc.
- **color** such as "red", "green", etc.
- **owner** such as "Amy Jones".
- **odometer** holds the number of miles traveled by the car total.
- **numRepairs** which is the number of repairs that have been made to the car since new.

Make members *protected.*

In addition, you will include the following *methods*:

- **sellTo()**- that takes one argument (a String), namely the name of the new owner of the car. This is how you sell a car.
- **Repair() -** with no parameters that increments the repair count by one when called.
- **Travel() -** increments the odometer by the number of miles passed to it. It takes one integer argument.
- **isReliable() -** a car is considered reliable if it has not been repaired more than twice per year on average since new. Assume that the current year is 2023.
- **isHighMilage()** a car is considered high milage if the average number of miles traveled per year is greater than 12000. This, of course, returns a Boolean.

When I name a method starting with ***is*** as in **isReliable**, that is my way of saying that it is a boolean function. That is, the return type of the method must be boolean.

Add getters and setters if needed. Do not add them for all fields! Think about the interface and design. For example, numRepairs is incremented by the **repair**() method (see above) so there is **no need** for a **setNumRepairs**().
Your class should also inherit (i.e., use the implements keyword) on the **Comparable** interface. Your compareTo() method should take an Object as a parameter. It will compare two fields and its ordering is as follows:

1. compare on the 'make' field. If one sorts before the other, then return compareTo() of the 'make' string field and you are done
2. if the 'make' fields compare as equal (returns 0), then return the compareTo() on the 'model' String fields. This will have the affect of sorting 'model' within 'make'.

As an example, if you have 4 cars with the following makes and models:

Make	**Model**
Ford	Taurus
Ford	Explorer
Chevy	Malibu
Chevy	Corvette

The cars should sort in the following order:

Chevy Corvette
Chevy Malibu
Ford Explorer
Ford Taurus

Also include at least one constructor that makes sense. Now, write a main method in this class that tests your implementation of Car. Assure that you test each of the methods you implemented, preferably with at least two (2) test cases for each method. Note that I am asking you to test your class in each individual classes for Problems 1, 2, and 3.

Turn in `Car.java`.

Problem 2 (10 points)

Starting with your Car class, design a class named `PassengerCar` in the file named `PassengerCar.java` as a subclass of `Car` implemented in Problem 1 above. The `PassengerCar` class should **extend** the Car class.

This class should include the following attributes (fields):

- `numPassengers` (an integer), that represents the number of passengers the car can carry.
- `numDoors` (an integer), the number of doors.
- `transmissionType` (a String), either ”automatic” or ”manual” transmission.

In addition, you will include the following methods:

- `isComfortable`() - a passenger car is considered comfortable if it can seat at least four people AND if it has at least four doors AND it is not older than seven years. (an unusual definition, but it will serve its purposes here).
- `isHardToDrive`() - a passenger car is considered hard to drive if its transmission type is manual.

Add getters if you feel they are needed. Again, thinking about the design, you will not need setters as once a car is built, the capacity, number of doors and transmission type are unlikely to change!

Also include at least one constructor that makes sense.
Remember that you are inheriting some or all (depending on how you designed your superclass) of the attributes and methods defined in its superclass (`Car`).
Now, write a main method in this class that tests your implementation of `PassengerCar`. Assure that you test each of the new methods you implemented, preferably at least 2 test cases each.

Turn in `PassengerCar.java`.

Problem 3 (5 Points)

Finally, design a class named `SportsCar` in the file named `SportsCar.java` as a subclass of `Car` implemented in Problem 1 above. The `SportsCar` class should **extend** the Car class.

This class should include the following attributes (fields):

- `maxSpeed` (an integer), that represents how fast the car can go.
- `seconds` (an integer), the number of seconds it would take to reach 100 miles per hour from 0.
- `isConvertible` (a Boolean): true or false.

In addition, you will include the following method:

- `isSnazzy`() a sports car is considered snazzy if it can drive faster than 150 miles per hour AND it is convertible, AND its color is red, purple, or yellow.

Add getters if you feel they are needed.

Also include at least one constructor that makes sense.

Remember that you are inheriting some or all (depending on how you designed your superclass) of the attributes

and methods defined in its superclass (`Car`).
Now, write a main method in this class that tests your implementation of `SportsCar`. Again, assure that you test each of the new methods you implemented, preferably with at least two (s) test cases each.

Turn in `SportsCar.java`.

Submission Instructions

Please follow this procedure for submission:

1. Place the deliverable files (`Car.java`, `PassengerCar.java` and `Sportscar.java`) into a folder by themselves. The folder's name should be CSE114_PS9_<yourname>_<yourid>. So if your name is Joe Cool and your id is 12345678, the folder should be named 'CSE114_PS9_JoeCool_12345678.

2. Compress the folder and submit the zip file.
    a. On windows, do this by pressing the right-mouse button while hovering over the folder. Select 'Send to -> Compressed (zipped) folder'. The compressed folder will have the same name with a .zip extension. You will upload that file to the Brightspace dropbox for Assignment9
    b. On mac, move the mouse over the folder then right-click (or for single button mouse, use Control-click) and select **Compress**. There should now be a file with the same name and a .zip extension. You will upload that file to the Brightspace dropbox for Assignment9.
3. Navigate to the course Brightspace site. Click **Assignments** in the top menu. Click **Assignment9** in the content area. Under **Submit Assignment**, click **Add a File**. Click **My Computer** at the top of the option list. Find the zip file in your file browser and drag it to the Upload area of the dialog box that appears. Click the **Add** button at the bottom left of that dialog.
4. ***Important!!*** Click the **Submit** button! If you fail to click **Submit**, the assignment will not be downloadable and will not be graded.