The State Universtiy of New York, Korea
Computer Science

CSE 114 Handout 14: PS 10 May 23, 2019

This problem set is due **Wednesday, May 29 at 11:59pm, KST**. Note that the due date that you see on Blackboard is not accurate since it shows the time in EST. You should go by the due date in this handout.

- Follow the specification exactly. It makes grading much easier.
- Each method that you write must be properly commented and attractively formatted.
- You may create and run your programs using Eclipse if you want, or continue using emacs and command line interface.
- Your programs should be formatted in a way thats 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.
- **Be sure to include** a comment at the top of each file submitted that gives **your name** and **email address**.

What to Submit

Submit the following files as a **single zip or tar file** on **Blackboard**. Multiple submissions are allowed before the due date. Please do **not** submit `.class` files, Eclipse-specific project files, or any that I did not ask for.

```
All the Java files that we will need to run your programs for
Problems 1 through 4.
```

What Java Features to Use

For this assignment, you are not allowed to use more advanced features than what we have covered in Lecture 1 through Lecture 20 (everything prior to recursion).

Partial vs. Complete Solutions

Please read what I said in PS 1 on this issue.

Naming Conventions In Java And Programming Style In General

Refer to the ones given in PS 1.

Problem 1 (50 points)

In this problem you will add more features to the `Shape` interface that we studied in class (Lecture 19, May 20) as described below.

Starting with the files in the interface folder of Lecture 19, first, add at least two more methods to the `Shape` interface.

Second, update `Circle.java` and `Rectangle.java` to be consistent with the changes that you made to `Shape.java`.

Third, add another class named `Triangle` in a file named `Triangle.java` that `implements Shape`. You should decide how you are going to represent a triangle. If you select the simplest kind of triangles, it will be acceptable.

Finally, include `UseShape.java` that uses `Circle`, `Rectangle`, and `Triangle`. Your new `UseShape.java` may be quite different from my original `UseShape.java` that we studied in class. You should decide what to include in this file to demonstrate that your new design (and implementation) is tested adequately. Hand in all the files that make up your solution for this problem.

Problem 2 (30 points)

In Problems 2 through 4, 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 2, 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. 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", etc.
- *year* such as 2019, 2007, etc.
- *color* such as "red", "green", etc.
- *owner* such as "Amy Jones".
- *numRepairs* which is the number of repairs that have been made to the car since new.

In addition, you will include the following methods:

- *sellTo* that consumes one argument, namely the name of the new owner of the car. This is how you sell a car.
- *repair* that increments the repair count by one when called.
- *isReliable:* a car is considered *reliable* if it has not been repaired more than once per year on average since new. Assume that the current year is 2019. 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.

Your class should also inherit (i.e., use the `implements` keyword) the `Comparable` interface, but I will let you decide whether to have this class inherit `Comparable` or its implementing subclasses inherit it directly themselves. This decision will be made based on what you want to compare as you try to compare two cars for greater than, less than, or equal to comparisons. At an appropriate place in your testing method, i.e., `main`, include testing with comparisons as well.

Also include at least one constructor that makes sense.

Now, write a `main` method in this class that tests your implementation of `Car`. Note that I am asking you to test your class in each individual classes for Problems 2, 3, and 4.

Problem 3 (30 points)

Now design a class named `PassengerCar` in the file named `PassengerCar.java` as a subclass of `Car` implemented in Problem 2 above.

This class should include the following attributes (fields):

- *numPassengers* that represents the number of passengers the car can carry.
- *numDoors*, the number of doors.
- *transmissionType:* 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 five people AND if it has at least four doors AND it is not older than five 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.

Also include at least one constructor that makes sense.

Don’t forget to implement the `Comparable` interface in this class if you chose to do it at this level.

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`.

Problem 4 (30 points)

Now design a class named `SportsCar` in the file named `SportsCar.java` as a subclass of `Car` implemented in Problem 2 above.

This class should include the following attributes (fields):

- *maxSpeed:* that represents how fast the car can go.
- *seconds:* the number of seconds it would take to reach 100 miles per hour from 0.
- *isConvertible:* `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.

Also include at least one constructor that makes sense.

Don’t forget to implement the `Comparable` interface in this class if you chose to do it at this level.

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`.