Claremont McKenna College
Computer Science

CS 62 PS 2 September 11, 2015

This problem set is due **11:55pm on Thursday, September 17**.

- Please follow the specification exactly. It makes grading much easier for us.
- Each method that you write must be properly commented and attractively formatted.
- Add your name and email address as comments at the top of each file you submit.

Submit the following files to the **Assignments** area on **Sakai**. Multiple submissions are allowed within the time limit. Please do **not** submit `.class` files or Eclipse project files.

```
Car.java
PassengerCar.java
SportsCar.java
Student.java
ProcessStudent.java
(Other files, if any, that are needed to run your program)
```

Introduction

In this problem set you will review some of the inheritance that you studied in CS 51 and abstract classes that we studied this semester. In addition you will practice using `ArrayLists` and file I/O.

Problem 1

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.

In Problem 1, we will first design `Car` in a file named `Car.java`. This class is to include attributes that are common in all these different kinds of cars, or cars in general. You will include the following attributes (fields) in the class `Car`:

- The *make* of the car such as "Chevy", "Lamborgini", "Toyota", etc.
- *model* such as "Outback", "Diablo", "Accord", etc.
- *year* such as 2015, 1999, etc.
- *color* such as "red", "green", etc.
- *owner* such as "Alex Johnson".
- *numRepairs* which is the number of repairs that have been done to the car since new.

In addition, you will include the following methods:

- *sellTo* that consumes one argument of type `String` that will be the next 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 so far. Assume that the current year is 2015.

Your class should also inherit (i.e., use the `implements` keyword) the `Comparable` interface. Let us also keep this class `abstract` by not implementing the `Comparable` interface in this class. That is, the `Comparable` interface will actually be implemented in the classes that inherit `Car`.

Also include at least one constructor that makes sense.

Since `Car` is an `abstract` class, it would not be possible to create instances of the class thus making testing difficult. You will test it in its subclasses.

Problem 2

Now design a class named `PassengerCar` in the file named `PassengerCar.java` as a subclass of `Car` implemented in Problem 1 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 five people AND if it has 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. Let us use the following rather strange definition in comparing two passenger cars: A passenger car is considered smaller (less) than another passenger car if the sum of the year it was made, the number of passengers it can carry, and the number of doors it has is smaller than the sum of the three same attributes of the other passenger car.

Now, write a `main` method that tests your `PassengerCar` class implementation. Make sure you include the attributes and methods defined in `Car` in your tests in this `main`.

Remember that you are *inheriting* the attributes and methods from its superclass (`Car`).

Problem 3

Now design a class named `SportsCar` in the file named `SportsCar.java` as a subclass of `Car` implemented in Problem 1 above.

This class should include the following attributes (fields):

- *maxSpeed:* represents how fast the car can go.
- *numSeconds:* the number of seconds to reach 100 miles per hour from start.
- *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 100 miles per hour AND it is a convertible, AND its color is red, pink, or yellow, and it can reach 100 MPH in 4 seconds or less.

Also include at least one constructor that makes sense.

Don't forget to implement the `Comparable` interface in this class. Let us again use the following rather strange definition in comparing two sports cars: A sports car is considered smaller (less) than another sports car if the sum of the year it was built and the maximum speed it can drive is smaller than the sum of the two same attributes of the other sports car.

Now, write a `main` method that tests your `SportsCar` class implementation. Make sure you include the attributes and methods defined in `Car` in your tests in this `main`.

Remember that you are *inheriting* the attributes and methods from its superclass (`Car`).

Problem 4

In this problem your program will read an input file containing student information into an `ArrayList` of student objects. Once you have them read into an `ArrayList`, you can do some interesting operations with the data that you just read in.

See a sample input file named `s0.txt` in the [given] folder on the web. I added multiple input files of various sizes for your consumption if you need them. An entry (a student record) in the input file has six attributes and is formatted as follows:

- The very first line of the input file contains a number indicating the number of records contained in the entire input file. You may use this number or ignore it if you choose to.
- Name attribute: the word `Name:` followed by first name, middle name (or middle initial with a period), and last name in that order in a single line, all separated by a blank space.
- Address attribute: the word `Address:` followed by street address in one line, city name in the next line, state in another line, and finally zip code in a separate line.
- Phone number attribute: the word `Phone:` followed by area code, prefix, and last four digits all separated by a blank space.
- ID attribute: the word `Id:` followed by an id number separated by a blank space in one line.
- Major attribute: the word `Major:` followed by a major, e.g., Computer Science, Economics, etc. A multi-word name should be allowed.
- GPA attribute: the word `GPA:` followed by a number, e.g., 3.76.
- Each record is separated by a blank line.
- The attributes in theory can appear in any order within a record, but you may assume that they appear in the order given in this format description.

Write a Java program (`ProcessStudent.java`) that reads in a file containing student records in the form described above into an `ArrayList` and includes the features described below:

- `averageGPA`: this method computes and returns the average GPA of all the students read in. Use a `for-each` loop in implementing this method.
- Add a method (say `main`) that calls `averageGPA` and prints the result to the standard output device. Make sure you got the correct result. Do this for each method that you will write below.
- `highestGPA`: this method takes the name of a state as a parameter and returns a student object whose GPA is the highest among the students from that state. It should return `null` if none found. Use an iterator in implementing this method.
- `histogram:` this method returns the number of students from each state as a `String` in the following format:

```
"State, No of Students
Arizona, 5
California, 20
Oregon, 6
Washington, 7"
```

  You may assume that the only valid state names are the 50 states in the USA. That is, you will only see at most those names in your input file. In the return value, do NOT include the state names from which there is no student.
- `studentsMajoringIn`: this method takes a major, e.g., `"Computer Science"`, and returns an array containing all the students who are majoring in that major.
- `studentsWithAreaCode`: this method takes an area code, e.g., `"909"`, and returns an `ArrayList` containing all the students who have that area code in the phone number.

Hand in the Java files (I assume you will at least have a class named `Student`?) that make up your solution and at least one input file that works with your program. I assume that your input file would be of the same format as mine, but I would still like to have one of your input files that for sure works with your program. The one that you hand in must have exactly 20 student records. Remember to update the count at the top of the file when you add more records.

Advice: When you test your program as you develop it, use a small input file, for example start with one line and make sure it works; then two lines and make sure it works, and so on. *Incremental development,* right? It is much easier to deal with a smaller input file. After you think you are done debugging your program, use a large input file to test it again before you hand it in.

Extra: I included a program that you can use to generate an input file of any arbitrary size. The file is called `RandomStudents.java`. Take a look if you are interested.