SUNY, Korea Computer Science

CSE 114 Handout 9: Problem Set 9

Due: Nov 29, 2019
Points: 30

Read This Right Away

This problem set is due at **11:59 pm on Nov 29, 2019, KST**. Don't go by the due date that you see on Blackboard because it is in EST. Go by the one given in this handout.

• To solve each problem below, you will be implementing a Java class.
• Please read carefully and follow the directions exactly for each problem. Files and classes should be named exactly as directed in the problem (including capitalization!) as this will help with grading.
• You should create your programs using emacs.
• 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.
• This problem set assumes you have installed Java and emacs on your computer. Please see the course web for installation instructions.
• You must use the command-line interface to run your programs. That is, you must use the `javac` and `java` commands to do this problem set. Do not use Eclipse yet.
• **Important!** You must use only 1 Scanner object. Assignments may require a Scanner object to be passed into other methods but do not create a new Scanner inside those methods! Use the one passed to the method. ***Creating multiple scanners causes a problem with the automated grading procedure and doing so will result in a loss of points on the assignment!***
• **Be sure to include** a comment at the top of each file submitted that gives **your name** and **email address**.
• **Remember:** We are now also assessing coding style. Be sure to use consistent indentation, good and informative variable names and write comments in your code to describe what it is doing.

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)

Suppose you are writing a program that will help our Admissions Office process the applications into the University. In this problem your program will open an input file containing applicant information and read it into an array of application objects. Once you have them read into an array, you can do some interesting operations with the data that you just read in. See a sample input file named `a0.txt` on the course web. I added multiple input files of various sizes you're your consumption if you need them. An entry (an application 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.
- The first line of an application record is the name of an applicant consisting of 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.
- Next line is the street address in one line, city name in the next line, state in another line, and finally zip code in a separate line.
- The next line is a phone number consisting of the area code, prefix, and last four digits all separated by a blank space.
- The next line is the id number of the application.
- The next line is the intended major, e.g., Computer Science, Economics, etc. A multi-word name should be allowed.
- The next line is the applicant's high school GPA, e.g., 3.96.
- The next line is an indication of whether the applicant is applying for a scholarship or not.
- If there is another application record in the file, it will follow next in the same format.

Your task is to read a file containing application records and do what is asked to be done in `ProcessApplications.java` that is provided in the [given] folder.

- Your solution should consist of the following files:
- `Application.java`: This file contains a class definition that represents an application.
- `ProcessApplications.java`: This file contains a `main` that reads in an input file and processes the application data. (A skeleton is provided in the [given] folder.)
- It is likely that you will want to add another file, perhaps `State.java` to represent a state and use it in the `main`. More on this in the FAQ.

Hand in your Java files 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.
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.
Note: I included a program that you can use to generate an input file of any arbitrary size. The file is called `RandomApplications.java`. Take a look if you are interested.

Hints: Read the FAQ!

Problem 2 (10 Points)

In Problems 2 through 4, you will be designing several classes. First we will design a class named `Jet`; then `BizJet` and `PassengerJet` each as a subclass of `Jet`. Conceivably you could design many more classes as subclasses to model other kinds of jets, for instance FighterJet's (e.g., F-18 Hornet), CargoJet (e.g., Lockheed C-5 Galaxy), etc., but we will not worry about them in this problem set. In Problem 2, we will first design a class named `Jet` in a file named `Jet.java`. This class is to include attributes that are common in all these different kinds of aircraft. You will include the following attributes (fields) in the `Jet` class:

- The *manufacturer* of a Jet such as "Boeing", "Lockheed", "McDonnell-Douglas", etc.
- *model* such as "747", "DC-10", "F-18", etc.
- *year* such as 2019, 2007, etc.

- *owner* (e.g., “Asiana”, “US Air Force”, “Rockwell Collins”, etc)
- *grossWeightEmpty* which is the weight of the aircraft in lbs
- *lastOverhaul* which is the number of flight hours since the last engine overhaul (assume all enginers are overhauled at the same time for multi engine aircraft.
- *numOverhauls:* This is the number of overhauls that have been performed on the engine
- *maxRecommendedFlightHours* which is the maximum recommended flight hours before performing an overhaul.

In addition, you will include the following methods:

- *sellTo* that consumes one argument, namely the name of the new owner of the jet. This is how you sell a jet.
- *overhaul* that resets the hours since last overhaul when called.
- *timeTillOverhaul:* This returns the number of flight hours left before reaching the maximum hours before overhauling the engines.
- *fly:* This takes 1 parameter, an integer number of hours that the plane has flown for a particular day. It increments *lastOverhaul* by that many hours.
- *needsOverhaul:* This returns a Boolean to indicate if the amount of flight hours is within 100 hours of the maximum recommended flight hours between overhauls.
- *isAging:* This returns a Boolean. True if the Jet is more than 15 years old and has had 20 overhauls or more. Otherwise, it returns false. Assume the current year is 2019.

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 jets 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.

In each of the problems (2,3, and 4) you may provide getter and setter classes but only the ones you think are essential.

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

Important note on testing! When you write test calls in your main, be sure to test all of the methods provided each in several cases. For example, when testing *needsOverhaul(),* be sure to create a jet that doesn’t need an overhaul and one that does. This will show the method is correctly analyzing the data.

Problem 3 (5 Points)

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

This class should include the following attributes (fields):

- *numPassengers* that represents the number of passengers the jet can carry.
- *numEngines:* The number of engines the Jet has
- *hasAutopilot:* The aircraft has full autopilot capabilities.

Additionally, write the following methods:

- *isHardToFly:* This returns a Boolean. The jet is hard to fly if there is no autopilot.
- *needsLongRunway:* This returns true if the grossEmptyWeight is greater than 500000 lbs.

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 (`Jet`).

Now, write a main method in this class that tests your implementation of `PassengerJet`.

Problem 4 (5 Points)

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

This class should include the following attributes (fields):

- *numPassengers:* that represents how many passengers the Jet can carry.
- *transOceanCertified:* This is a Boolean that indicates the aircraft is certified to fly over oceans by the FAA

In addition, you will include the following method:

- *theJetRocks:* This returns true if the Biz Jet 'rocks' (is awesome). The jet rocks if it holds at least 8 passengers and is *transOceanCertified*.

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 (`Jet`).

Now, write a `main` method in this class that tests your implementation of `BizJet`.

Submission Instructions

Please follow this procedure for submission:

1. Place the deliverable files (`Application.java, ProcessApplications.java, State.java` [if you implemented State], `Jet.java`, `PassengerJet.java` and `BizJet.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 Blackboard dropbox for PS9.
    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 Blackboard dropbox for PS9.
3. Navigate to the course blackboard site. Click **Assignments** in the left column menu. Click **PS9** in the content area. Under **ASSIGNMENT SUBMISSION**, click **Browse My Computer** next to **Attach Files.** Find the zip file and click it to upload it to the web site.
4. ***Important!!*** Click the **Submit** button! If you fail to click **Submit**, the assignment will not be downloadable and will not be graded.