CSE 114 Fall 2023: Assignment 10

Due: December 4, 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.

Important Note: For this one assignment, I am distributing it just ahead of the class on Recursion. You can start the first problem which is on Abstract Classes and start the remaining problem after the Recursion class.

Problem 1 (15 points)

In this problem, you will be designing several classes. First a class named `Jet`; then `BusinessJet` 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.

Part 1:

First design a class named `Jet` in a file named `Jet.java`. This to be an abstract class (so we cannot instantiate it!) and 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 the Jet (a String) such as "Boeing", "Lockheed", "McDonnell-Douglas", etc.
- *`Model`* `(a String)` such as "747", "DC-10", "F-18", etc.
- *`year`* built (an integer), such as 2019, 2007, etc.
- *`owner`* (a String) (e.g., "Asiana", "US Air Force", "Rockwell Collins", etc)
- *`grossWeightEmpty`* (an integer) which is the weight of the aircraft in kilograms
- *`hardLandings`* (an integer) which will count the number of hard landings the plane has undergone.
- *`lastOverhaul`* (an integer) which is the number of flight hours since the last engine overhaul (assume all engines are overhauled at the same time for aircraft with multiple engines)
- *`numOverhauls`* (an integer): This is the number of overhauls that have been performed on the engine
- *`maxRecommendedFlightHours`* (an integer) which is the maximum recommended flight hours before performing an overhaul.

The constructor should take parameters matching the initial value of each of the members in the order given above!

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 the last engine overhaul when called. It should also zero out the hard landing count! This takes no arguments and returns void.
- *`hardLanding:`* which adds one to the **hardLandings** member (imagine this is triggered by a sensor on the landing gear of the aircraft.) The method takes no arguments.
- *fly:* This takes 1 parameter, an integer number of hours that the plane has flown for a particular day. It increments the *`lastOverhaul`* member by that many hours.
- *`isAging:`* This returns a Boolean. It returns true if the Jet is more than 15 years old **and** has had 25 or more overhauls. Otherwise, it returns false. Use the **LocalDateTime** class to determine the current year.

Additionally, the following two methods should be marked ***abstract*** and have no implementation in the Jet class.

- *`timeTillOverhaul:`* This returns the number of flight hours left before reaching the maximum hours before overhauling the engines. The maximum hours between overhauls has a base value but each jet has rules that sometimes modify this so the code is NOT implemented in Jet but will be in the subclasses. It takes no arguments and returns an integer.
- *`needsOverhaul:`* This returns a Boolean. It should be true if an overhaul is needed. The criteria for this varies based on the type of jet so it is not implemented here but will be in the subclasses.

Your class also implements (i.e., use the `implements` keyword) the `Comparable` interface. Thus you should have an implementation for **compareTo()** that compares based on **grossWeightEmpty**.

For all of the classes you build in this problem, you should provide getter and setter methods only on variables for which you feel they are needed.

Turn in **Jet.java**

Important note on testing! I will provide a main() method to test code for each of the two derived classes. For this class, however, it makes no sense since it is abstract and we cannot instantiate it.

Part 2

Now design a class named `PassengerJet` in the file named `PassengerJet.java` as a subclass of `Jet` implemented in Part 1.

This class should include the following attributes (members):

- *`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.
- *`ohAdjust`:* An object with type OverhaulHoursAdjust() This describes how to adjust the time between overhauls given the number of hard landings. I will provide the class with this type for you.
- *`maxHardLandings`:* The most hard landings the plane can take before the engines are required to be overhauled regardless of hours flown.

The constructor for PassengerJet should include all of the parameters needed for the Jet superclass plus the items in the above list here **in that order!**

Additionally, write the following methods:

- *`isDifficultToFly`*: This returns a Boolean. The jet is hard to fly if there is no autopilot. It takes no arguments.
- *`needsLongRunway`:* This returns true if the *`grossWeightEmpty`* is greater than 300,000 kilograms. It takes no arguments.

Now, you will also need to implement the **timeTillOverhaul()** method since it is abstract in **Jet**.. For a `PassengerJet` (and for `BizJet` as well), the **maximumNumberOfFlightHours** must be adjusted as the aircraft is subjected to hard landings. One of the parameters to the constructor is an **OverhaulHoursAdjust** object which will contain the information required to adjust the **maximumNumberOfFlightHours**. The two values in **OverhaulHoursAdjust** are a number of years and an adjustment percentage. If the aircraft is older than the number of years given, reduce the **maximumNumberOfFlightHours** by the given percentage.

As an example, if a PassengerJet is created with OverhaulHoursAdjust(10, 5) as the ohAdjust parameter, then when you compute timeTillOverhaul(), if the page of the aircraft is greater than or equal to 5 years, you first reduce **maximumNumberOfFlightHours** by 10 percent (i.e. if it was 2000 hours, then it is now 1800 hours). You then subtract hours flown to get the result you need to return.

You can determine age by getting the current year from a LocalDateTime object (look this up in the API docs!).

Finally, implement the **needsOverhaul()** method which is abstract in **Jet**. This returns a Boolean (true/false) if it is time to overhaul the engines. For PassengerJet, it should simply return true if the aircraft is within 100 hours of the modified maximum flight hours. In addition, if the maximum number of hard landings is exceeded, it should just return true without regarding hours flown.

I will provide a main method to test the code. Be sure that you follow the above guidelines for the constructor as I will create specific aircraft and call specific methods to test your code. The test code will be in the provided PassengerJet.java file.

Sample run for PassengerJet with given test main

If you run your code with the given main() method I supply in `PassengerJet.java`, your code should print something close to the following

```
Testing PassengerJet ==========
J1 is greater than j2
j1 time till overhaul:-90.0
J1 needs overhaul? true
Add 30 flying hours j1
j1 time till overhaul:-120.0
J1 needs overhaul? true
J1 do the overhaul
j1 time till overhaul:1800.0
J1 needs overhaul? false
J2 needs overhaul? false
j2 time till overhaul:300.0
Add 60 flying hours j2
j2 time till overhaul:240.0
J2 needs overhaul? false
12 hard landings for j2
j2 time till overhaul:240.0
J2 needs overhaul? true
J1 is aging! false
J2 is aging! true
J1 is hard to fly! false
J2 is hard to fly! true
J1 needs long runway! true
J2 needs long runway! true
```

Turn in PassengerJet.java

Part 3

Now design a class named `BizJet` in the file named `BizJet.java` as a subclass of the `Jet` implemented in Part 1.

This class should include the following attributes (members):

- *`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

Like PassengerJet, take the parameter list from the Jet constructor, add initializers for the two items above and also add the OverhaulHoursAdjust object and a maximum hard landings initializer.

In addition, you will include the following method:

- *`theJetRocks`:* This returns true if the `BusinessJet` is considered high class jet. The jet is considered high class if it holds at least 12 passengers and is *`transOceanCertified`*.

For implementing **timeBeforeOverhaul(),** implement code similar to that of PassengerJet. However, also add a check and if the jet is less than or equal to 3 years, add 50 hours to the **maximumNumberOfFlightHours** before computing the time before the next overhaul.
Finally, override the **needsOverhaul**() method. This returns a Boolean (true/false) if it is time to overhaul the engines. For `BizJet`, it should simply return true if the aircraft is within 100 hours of the modified maximum flight hours. In addition, if the maximum number of hard landings is exceeded, it should just return true without regarding hours flown.

Sample run for BizJet with given test main

If you run your code with the given main() method I supply in `BizJet.java`, your code should print something close to the following:

```
Testing BizJet ============
J1 is greater than j2
j1 needs overhaul? true
j1 time till overhaul:10.0
j1 time till overhaul:-20.0
j1 needs overhaul? true
j3 needs overhaul? false
j3 time till overhaul:160.0
Add 60 hours to j3 flight time
j3 time till overhaul:100.0
J3 needs overhaul? false
J3 do overhaul!
j3 time till overhaul:2050.0
J3 needs overhaul? false
6 hard landings for j3
j3 time till overhaul:2050.0
J3 needs overhaul? false
J2 needs overhaul? false
j2 time till overhaul:300.0
j2 time till overhaul:270.0
J2 needs overhaul? false
J1 is aging! false
J2 is aging! false
J1 does not rock!
J2 does not rock!
```

Turn in BizJet.java

Problem 2 (10 points)

Now, write a class named `Recursion.java`. This will contain solutions to the following problems. The solutions to each of the following problems ***MUST*** be recursive or no credit will be given for that problem.

Part 1

You saw a procedure for how to convert decimal numbers to binary. Now, write a recursive method, d2b() that takes an integer argument and returns a string of binary digits. The method header should match:

```
public static String d2b(int value);
```

Part 2

Write a recursive method that takes an array of integers and returns the product of all integers in the array (the result of multiplying them all together). Hints: This may be a good place to use an auxiliary method with extra parameters. Also, think carefully about the base case!

The method header should look like this:

```
public static int arrayProduct(int[] numbers);
```

Parts 1 and 2 should go in a file called Recur.java (holding the class Recur, the recursive methods, and a short main to make a couple test calls).

Turn in Recur.java.

Part 3 [Extra credit! 5 points]

Write a recursive method that prints a solution to the tower of hanoi problem. Place your solution in a separate Java source file called TowerOfHanoi.java. Write a main method with a couple calls to your method that solves the problem.

Here is how the classic problem is presented:

You have 3 posts. 1 Post has a stack of discs. The discs are in order of size from smallest at the top to largest at the bottom. You must move the discs from post 1 to post 3. The rules:

You may use the middle post has a 'temporary' or work post

You can only move one disc at a time.

You can never place a larger disc on a smaller disc.

Here is a more detailed description of the task:

https://en.wikipedia.org/wiki/Tower_of_Hanoi

If you think this through, it is not too hard. The base case is when you have 1 disc. Just move it to the target post. With 2 discs, you have to move the top disc o the working post, then the next disc to the target post, and finally, the disc on the working post to the target.

To show the disc movements for the solution, messages should be printed similar to:

Move 1 -> 3

Which indicates moving the top disk on post 1 to post 3.

Thus, running the method with the argument 3 would print:

```
Move: 1 -> 3
Move: 1 -> 2
Move: 3 -> 2
Move: 1 -> 3
Move: 2 -> 1
Move: 2 -> 3
Move: 1 -> 3
```

Think about how you have to specify the recursive case (each call of this 'tower of hanoi' function could call itself more than once!)

Again, I recommend using an auxiliary method with extra parameters to help organize what each call of this method is doing.

Turn in **TowerOfHanoi.java**

Submission Instructions

Please follow this procedure for submission:

1. Place the deliverable files (`Jet.java, PassengerJet.java, BizJet.java,` and `Recur.java`. [Also, `TowerOfHanoi.java` if you do the extra credit!]) into a folder by themselves. The folder's name should be CSE114_PS10_<yourname>_<yourid>. So if your name is Joe Cool and your id is 12345678, the folder should be named 'CSE114_PS10_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 Assignment10
    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 Assignment10.
3. Navigate to the course Brightspace site. Click **Assignments** in the top menu. Click **Assignment10** 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.