CSCE 314 [Sections 595, 596, 597] Programming Languages – Spring 2024

Hyunyoung Lee

Homework Assignment 4

Assigned on Wednesday, March 6, 2024

Electronic submission to Canvas due at **11:59 p.m. on Friday, 3/22/24**

By electronically submitting this assignment to Canvas by logging in to your account, you are signing electronically on the following Aggie Honor Code:

"On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment."

In this assignment, you will practice some basics of object-oriented programming in Java. The assigned problems are either directly from our Java textbook, a slight variation of the textbook exercise problems, or based on the codes that are explained in the textbook.

You will earn total 130 points. Here are some general instructions.

1. This homework set is an *individual* homework, not a team-based effort. Discussion of the concept is encouraged, but actual write-up of the solutions must be done individually by yourself. Your final product – the code as well as comments, explanations, the README file – *should never be shared*.

2. Read the problem descriptions and requirements carefully! There may be significant penalties for not fulfilling the requirements.

3. **Explain each line or block of your code in your own words in the comments.** Even though you think the code is self-explanatory, explain in your own words anyway! Your work will be graded not only on the correctness of your code, but also on the consistency and clarity with which you express it.

4. Turn in one *yourFirstName-yourLastName-hw4.zip* file on Canvas, nothing else. Your zip directory must include the two .java files (Fibonacci.java and Vehicle.java) with your implementations in them, and a README file that explains how to compile and execute your codes, and what is the expected output of your codes when tested. **The README file is worth ten points.**

5. All Java code that you submit must compile without error using `javac` (at the terminal) of Java version 11[1] or higher (the most recent version is Java 21). If your code does not compile using javac, you risk receiving zero points for this assignment.

6. Remember to put the head comment in *all* of your files, including your name, your UIN, and *acknowledgements of any help received* in doing this assignment. Again, **remember the honor code!**

Problem 1. (10 points) Explain in a README file (plain .txt file) how to compile and execute your codes, and what is the expected output of your codes when tested (clearly marked for each problem). It should be detailed enough so that whoever grades your codes can understand what you were doing with your code clearly and should be able to reproduce your tests following what you wrote in it.

[1] Java 11 is installed in the departmental servers, linux2.engr.tamu.edu and compute.engr.tamu.edu

Problem 2. (10 points) Section 1.6. Modify `ImprovedFibonacci` on pages 9–10 as instructed below. Name your modified class `SubsetOutputFib`, and place it in a file named *Fibonacci.java*.
Let f_n denote the n-th Fibonacci number. The `SubsetOutputFib` will accept two integer values as command line input, assign the first one to `be` (meaning begin) and the second one to `en` (meaning end), and print out only those Fibonacci numbers from $f_{\texttt{be}}$ to $f_{\texttt{en}}$. For example, if the two command line arguments are given as 4 and 7 in this order, then the output should be:

```
4:  3
5:  5
6:  8 *
7:  13
```

Make sure that you do the error checking whether both `be` and `en` are positive integers, and `be` $\leq$ `en`. In the case of an erroneous input, do not just quit the process but try to correct the error such that negative integers will be converted to their absolute values; and if `be` $>$ `en`, then swap the values of `be` and `en` so that `be` $\leq$ `en`. Then, inform the user about it (why their input was erroneous and how the program corrected it) and output the Fibonacci numbers of the corrected range.

Problem 3. (10 points) Section 1.9, Exercise 1.10 on page 20. Modify the `ImprovedFibonacci` on pages 9–10 (not the modified version in Problem 2). Put your code in the file *Fibonacci.java*.

Problem 4. (20 points) Section 2.1, Exercise 2.1 on page 44, Section 2.2, Exercise 2.3 on page 46, and Section 2.6, Exercise 2.13 on page 68. Work in a file Vehicle.java.
Use the following types for the fields: `int` for current speed, `int` for current direction in degrees (consider straight north as 0 degrees and the degree increments clockwise up to 359 degrees, thus for example, straight east is 90 degrees, straight south 180 degrees, and straight west 270 degrees), and `String` for owner name.
For Exercise 2.3, use `int` for both of the vehicle ID number fields. The `static` field for the next vehicle ID number should be simply incremented by one each time a vehicle instance is created.

Problem 5. (10 points) Section 2.5, Exercise 2.7 on page 54, and Section 2.6, Exercise 2.9 on page 58. You will continue working on your `Vehicle` class from Problem 4, with the `main` method in a different class named `VehicleTest`. In the main method, you will create ten vehicles using the constructors you added to the `Vehicle` class, first five of which using the no-arg constructor (setting the values using the setter methods that you implemented in Exercise 2.13) and the latter five vehicles using the one-arg constructor.
From now on, whenever you add new functionalities to `Vehicle`, you will add the test code for the new functionalities in the main method of `VehicleTest`.

Problem 6. (10 points) Section 2.6, Exercise 2.10 on page 60. Add test code of your `toString` method in the main method of `VehicleTest`.

Problem 7. (10 points) Section 2.6, Exercise 2.15 on page 68. Add test code of those methods in the main method of `VehicleTest`.

Problem 8. (10 points) Section 2.8, Exercise 2.17 on page 71. Add test code of those methods in the main method of `VehicleTest`.

Problem 9. (40 points) Section 3.1, Exercise 3.1 on page 79. Continue working in the file Vehicle.java. Add constructors and `toString` method for `PassengerVehicle` appropriately. Also, have `PassengerVehicle` implement `Comparable<PassengerVehicle>` and give the implementation of the `compareTo` method where the comparison criterion is the total number of seats.
In the main, create at least five instances of `PassengerVehicle` with varying total number of seats, sort them (using any sorting algorithm which will sort the `PassengerVehicle`s according to the `compareTo` implementation), and outputs them in a *descending* order.
Also, after some of the seats are occupied in each passenger vehicle, output the number of seats currently available in each vehicle.

Have fun!