CSCE 314 [Sections 500, 501] Programming Languages – Spring 2020

Hyunyoung Lee

Assignment 5

Assigned on Tuesday, March 24, 2020

Electronic submission to eCampus due **10:00 p.m., Friday, April 3, 2020**

By electronically submitting this assignment to eCampus 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 100 points.

Note 1: This homework set is **individual** homework, not a team-based effort. Discussion of the concept is encouraged, but actual write-up of the solutions must be done individually.

Note 2: Turn in one `yourLastName-yourFirstName-hw5.zip` file on eCampus, 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.**

Note 3: All Java code that you submit must compile without errors using `javac` of Java version 8 or higher (most recent version is version 14). If your code does not compile, you will likely receive zero points for this assignment.

Note 4: 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 (just .txt file) how to compile and execute your codes, and what is the expected output of your codes when tested. 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.

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`.

Problem 3. (10 points) Section 1.10, Exercise 1.13 on page 24. Modify the `ImprovedFibonacci` on pages 9–10 (not the modified version in Problem 2). Place the modified `ImprovedFibonacci` implementation in the same file as Problem 2 (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 2.4, Exercise 2.5 on page 50. Write the main method within a new class named `VehicleTestP4`. Create 5 vehicles and print their field values. To do so, create the vehicles (using the default constructor since you have not implemented any constructor of your own yet), set the values using the accessor (setter) methods that you implemented in Exercise 2.13 in the previous problem, and use the getter methods to print their field values.

Problem 6. (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, but the modified `main` needs to be in a different class, named `VehicleTest` since in the new main, you will now create the five vehicles using the constructor you added to the `Vehicle` class. Keep both `VehicleTestP4` and `VehicleTest` in the same file Vehicle.java. 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 7. (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 8. (10 points) Section 2.6, Exercise 2.15 on page 68. Add test code of those methods in the main method of `VehicleTest`.

Problem 9. (10 points) Section 2.8, Exercise 2.17 on page 71. Add test code of those methods in the main method of `VehicleTest`.

Have fun!