Module 2 Lab -- Automatic Transmissions

Due Sep 19 by 11:59pm **Points** 100

> **Goals:** To practice designing, implementing, and testing solutions using proper OO methodologies.

Automatic Transmissions

Cars with a manual transmission require the driver to change gears as they are changing speed. Putting it simply, this changes how much of the rotation of the engine affects the rotation of the axles (low gears transfer rotation slowly, so that even if the engine is revved up the car does not speed up much). This is the same as how gears on a bicycle work.

Cars with automatic transmissions solve this problem by letting the car computer choose how much rotation to transfer at certain speeds. Since cars cannot have negative speed (but can go as fast as their engines allow), this choice of gear is a monotonically increasing function of speed.

In this lab, you will be simulating this in a computer program. Vehicles cannot have a negative speed, but can go as fast as needed.

What to do

Package: transmission

Design an interface called `Transmission` that represents a single car transmission. Then implement this interface in a class called `AutomaticTransmission` which automatically determines the current gear by the current speed of the car. Your implementation should include:

- A constructor that takes 5 speed thresholds for the 6 possible gears in order (speed to go from 1 to 2 or back, speed to go from 2 to 3 or back, etc.). The constructor should ensure that the input values are valid and throw an `IllegalArgumentException` if any of the parameter values are not legal.
- A method called `increaseSpeed` which returns a `Transmission` object with speed increased by 2 and the appropriate gear.
- A method called `decreaseSpeed` which returns a `Transmission` object with speed decreased by 2 and the appropriate gear. This method should throw an `IllegalStateException` if the speed becomes invalid.
- Methods that get the speed and the current gear. These are accessors (a.k.a., *getters*), so they should be named appropriately.
- A `toString` method that can be used to get the current state of the transmission (speed and gear), as follows:

```
Transmission (speed = 45, gear = 3)
Transmission (speed = 0, gear = 1)
```

Testing

Whenever you write a class, you should also write tests for that class that prove not only that your code CAN work but also that it WILL ALWAYS work. Your goal here is to write tests to achieve as close to 100% coverage as possible. But even more importantly, your tests should be sufficient to ***convince someone else that your code works correctly***.

What to Submit

1. Create a zip file that directly contains only your src/ and test/ folders. When you unzip the file, you must see only these two folders.
2. Log on to the **Handins server** **(https://handins.ccs.neu.edu/)**
3. Navigate to this lab and submit the zip file.
4. Wait for a few minutes for the grader's feedback to appear, and take action if needed.

Grading Criteria

This assignment will be assessed via the automatic testing available on the Handins server. You should ensure that your code does not have any code style violations and that it passes all of the test cases. This is worth 1% of your final grade.

Additionally, your code will be assess via peer evaluation. Each student will be required to review one other student's code for an additional 0.5% of their final grade. This grade will be based on the quality of the review **you give**, not the review you receive. During the peer review your code will be evaluated based on three criteria:

1. Design and organization of classes, interfaces, and methods including
    - How appropriately you captured the data and operations in the implementation
    - Whether code looks well-structured and clean (not unnecessarily complicating things or using unwieldy logic)
    - How well your design and implementation embraces various design principles that you have learned so far
2. Understandability and quality of documentation including
    - Whether the documentation enhances the understandability of the code rather than just repeating how something is implemented
    - Whether it is stand alone, meaning whether the documentation is sufficient for use without reading the implementation

3. Quality and coverage of tests including
    - Whether tests are modular -- do you write one big, monolithic test method or does your test class have separate methods for each test
    - Whether tests are written in such a way so that the reader knows what is being tested
    - Whether the tests convince the reader that if all tests pass then the code it is testing works correctly