TypeScript Exercise

Task: This is an extension of Assessment 2. Instead of using JavaScript classes, you will first define an interface, and then build classes based on that interface. Your tests from the Car class should still pass.

Build Specifications:

- Create a file called `IVehicle.ts`, which will contain the interface . The interface will have the required properties and methods
    - Properties:
        1. `speed`
        2. `engineOn`
        3. `fuelLevel`
    - Methods
        - `accelerate`
        - `brake`
        - `turnCarOn`
        - `turnCarOff`
        - `refillFuel`
        -

`Remember that an interface only defines what the class should have, not any of the actual implementation.`

- Declare a class named `Car inside of Car.ts` with a constructor that accepts one parameter named `fuelLevel`. Give the parameter `fuelLevel` a default value of `100` if no argument is supplied.
    - Properties
        4. `speed` - initialized to the value of `0`
        5. `engineOn` - initialized to the value of `false`
        6. `fuelLevel` - initialized to the `fuelLevel` parameter
    - Methods
        1. `accelerate` - if `fuelLevel` is more than or equal to `1` then decrease `fuelLevel` by `1` and increase `speed` by `1`
        2. `brake` - decreases `speed` by `1`. `speed` cannot go below 0.
        3. `turnCarOn` - sets `engineOn` to `true`
        4. `turnCarOff` - sets `engineOn` to `false`
        5. `refillFuel` - sets `fuelLevel` to `100`

- Create an instance of `Car` called `myCar` with `60` as the argument for `fuelLevel`. Call the following methods in order. (You may also include any console.log's you want.)
    - refillFuel
    - turnCarOn
    - accelerate
    - accelerate
    - accelerate
    - brake
    - brake
    - brake
    - turnCarOff
- Compile your TypeScript files (run tsc). Run the tests from the assessment to make sure that they all work properly.
- Add the following classes: **Plane, Truck, Submarine.** Note that they all have some shared functionality, but they should also have their own unique properties.
- What are ways you can refactor this code to be easier to extend? One example: consider renaming turnCarOff() to turnOff() to be more generic and so that the naming makes more sense for any IVehicle type.

TESTS

Overview: This challenge contains 10 tests. The tests are used to check the accuracy of your assessment. This is a fully automated process. Below you will find a description of each test and what is being checked.

Test Cases:

- defaults fuelLevel to 100, defaults speed to 0, and defaults engineOn to false
- accepts an argument for fuelLevel
- increases speed by 1 when accelerate is called
- decreases fuelLevel by 1 when accelerate is called
- does not increase speed when fuelLevel is less than 1 and accelerate is called
- decreases speed by 1 when the brake method is called
- does not decrease speed when speed is equal to 0 and brake is called
- sets engineOn to true when turnCarOn is called AND sets engineOn to false when turnCarOff is called
- sets fuelLevel to 100 when refillFuel is called
- myCar has a final state of false for engineOn AND has a final state of 0 for speed