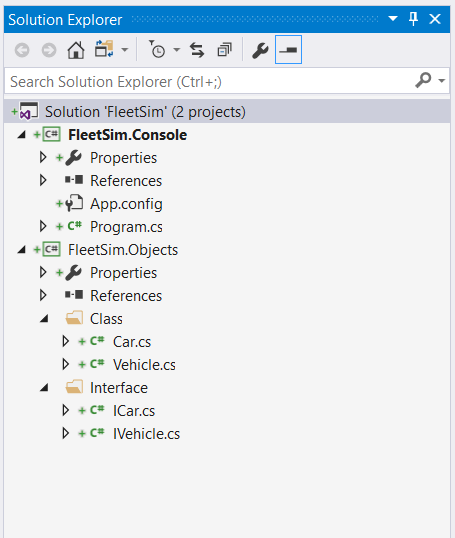
Objective: Create a simulator for fleet vehicle maintenance

Create a solution called “FleetSim” with the following structure



**Part 1: Create a class library Project called (FleetSim.Objects)**

Create a folder called “Interface” to contain all interfaces.

Create a folder called “Class” to contain all classes.

Create a folder called “Enum” to contain all enum types.

Create a “vehicle” base class with the following private fields, and public properties for accessing them.

1. VIN (random GUID)
2. Mileage
3. Color (red, white, blue using enum)
4. LastServicedDate
5. LastServicedMileage

Create a “car” class that inherits from “vehicle” with the following additional properties:

1. Year
2. Make
3. Model (i.e. Mustang, Focus, Explorer, etc.)

Add an overload constructor that allows you to set Year, Make, and Model.

Add another overload constructor that allows you to set Year, Make, Model, and color

Every Vehicle must implement the following methods via an interface

1. TuneUp()

{

If (Mileage - LastServiceMileage >= 30K)

{

Update LastServiceMileage & LastServicedDate

Console.WriteLine(“Vehicle tuned up at” + LastServiceMileage & LastServicedDate)

}

if (mileage>=100,000)

RebuildEngine();

}

1. RebuildEngine()

{

Mileage=0;

LastServiceMileage=0;

Console.WriteLine(“Engine rebuilt on ” + LastServicedDate for {model} );

}

**Part 2: Create a console project called FleetSim.Console**

Create a collection of 5 different vehicles with varying model names and colors.

All cars will start new (mileage=0, LastServiceMileage=0)

Loop 10 times

{

Foreach (car in collection)

{

Generate a random mileage (5-15K)

Add random mileage to each vehicle in the collection

Every time the mileage is updated, TuneUp() should be called to check the following:

1. If LastServiceMileage exceeds 30K miles, the update LastServiceMileage and LastServiceDate.
2. If vehicle exceeds 100K miles, it must call RebuildEngine() where it will update mileage=0 and LastServiceMileage=0.

}

}

Please remember:

* All data access must be done through properties or constructors
* Interfaces belong in separate folders, with each interface in a separate file
* Every class inherits from an interface.
* Classes belong in separate folders, with each class in a separate file.
* Use proper names for files and variables that would be meaningful to someone else having to modify your code.
* Structure your code into proper size functions. Each function should do a single task matching its name (action/verb).
* Use proper spacing and indentation so blocks of code are easily recognizable.
* Output must be properly formatted and appear professional.
* Use Try-Catch in the console project only.

Submission:

1. Due April 12, 2018 9:00 AM
2. Create “FleetSim” repository in GitHub where all project files will be stored.
3. Email a publicly accessible GitHub link to [samnasr@live.com](mailto:samnasr@live.com) with “FleetSim” in the subject line. In addition, send a screen caption of the test cases and the results.