Exercise 3 – Garage Management System

Objectives

- Integration of Classes, Inheritance and Polymorphism
- Working with Arrays / Collections / Data Structures
- Using Enums
- Development and use of external DII (Assembly)
- Working with numerous projects
- Working with Exceptions

Prior Knowledge

- Object Oriented Programming whilst using Polymorphism in C#
- Working with Arrays / Collections / Data Structures
- Development and use of external DII (Assembly)
- Working with numerous projects
- Working with Exceptions
- Link to a video tutorial about Exceptions (Watch first): https://www.youtube.com/watch?v=AWuUMHhLLmU
- Link to a video tutorial about inheriting from object: https://www.youtube.com/watch?v=bScwvq5-ovg

Exercise

Final objective – Developing a computer software that "manages" a vehicle garage. The system will manage a garage that handles these five types of vehicles -

- **Fuel-Based Motorcycle**

  2 tires with max air pressure of 30 (psi), Octane 96 (fuel), 6 liters fuel tank
- **Electric Motorcycle**

  2 tires with max air pressure of 30 (psi), Max battery life – 1.8 hours
- **Fuel-Based Car**

  4 tires with max air pressure of 32 (psi), Octane 98 fuel, 45 liter fuel tank
- **Electric Car**

  4 tires with max air pressure of 32 (psi), Max battery life – 3.2 hours
- **Fuel-Based Truck**

  12 tires with max air pressure of 28 (psi), Octane 96 fuel, 115 liter fuel tank

<u>Every</u> vehicle contains the following properties:

- Model Name (String)
- License Number (String)
- Remaining Energy Percentage (Fuel/Battery) (float)
- Wheels, with each wheel containing the following:
  - Manufacturer Name (String)
  - Current Air Pressure (float)
  - Max Air Pressure Recommended by the manufacturer's (float)
  - Inflate Action (A method that receives how much more air <u>to add</u> to a wheel, and changes the air pressure while not crossing the max limit)
- **A motorcycle (Fuel/Electric), apart from the vehicle properties, also has the following properties:**
  - License type – A, A1, B1, B2
  - Engine Volume (int)

- **A car (Fuel/Electric), apart from the vehicle properties, also has the following properties:**
  - Color – Gray, Blue, White, Black
  - Number of doors – 2, 3, 4, or 5
- **A truck, apart from the vehicle properties, also has the following properties:**
  - Is cooled? (bool)
  - Volume of cargo (float)
- **Every fuel based vehicle contains the following properties and operations:**
  - Fuel type – Soler, Octane 95, Octane 96, Octane 98
  - Current amount of fuel in liters (float)
  - Max amount of fuel in liters (float)
  - Refueling operation - (A method that receives how much more fuel to add, and changes the amount of fuel, if the fuel type is correct, and the fuel tank is less than full)
- **Every electric based vehicle contains the following properties and operations:**
  - Remaining time of engine operation in hours (float)
  - Max time of engine operation in hours (float)
  - Recharge operation - (A method that receives how many more hours to add, and charges the battery, while not crossing the max limit)
- **Every individual vehicle in the garage contains the following properties:**
  - Owner name (String)
  - Owner Phone Number (String)
  - Vehicle status – In Repair, Repaired, Payed for
    - Every new vehicle is initiated as “In Repair”

The system will supply the user with the following functions:

1. “Insert” a new vehicle into the garage. The user will be asked to select a vehicle type out of the supported vehicle types, and to input the license number of the vehicle. If the vehicle is already in the garage (based on license number) the system will display an appropriate message and will use the vehicle in the garage (and will change the vehicle status to “In Repair”), if not, a new object of that vehicle type will be created and the user will be prompted to input the values for the properties of his vehicle, according to the type of vehicle he wishes to add.
2. Display a list of license numbers currently in the garage, with a filtering option based on the status of each vehicle
3. Change a certain vehicle’s status (Prompting the user for the license number and new desired status)
4. Inflate tires to maximum (Prompting the user for the license number)
5. Refuel a fuel-based vehicle (Prompting the user for the license number, fuel type and amount to fill)
6. Charge an electric-based vehicle (Prompting the user for the license number and number of minutes to charge)
7. Display vehicle information (License number, Model name, Owner name, Status in garage, Tire specifications (manufacturer and air pressure), Fuel status + Fuel type / Battery status, other relevant information based on vehicle type)

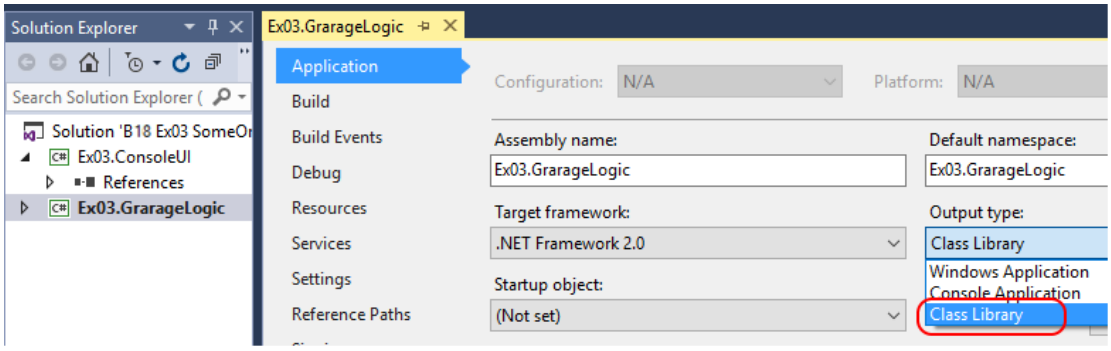
When developing the system, create a solution which contains **two** projects:

1. **Ex03.GarageLogic**

   This project creates a .dll file which contains only the object model and the logical layer of the system (the layer we will want to reuse if we will develop an alternative user interface in the future).

   This layer cannot prompt the user or receive data from the user.

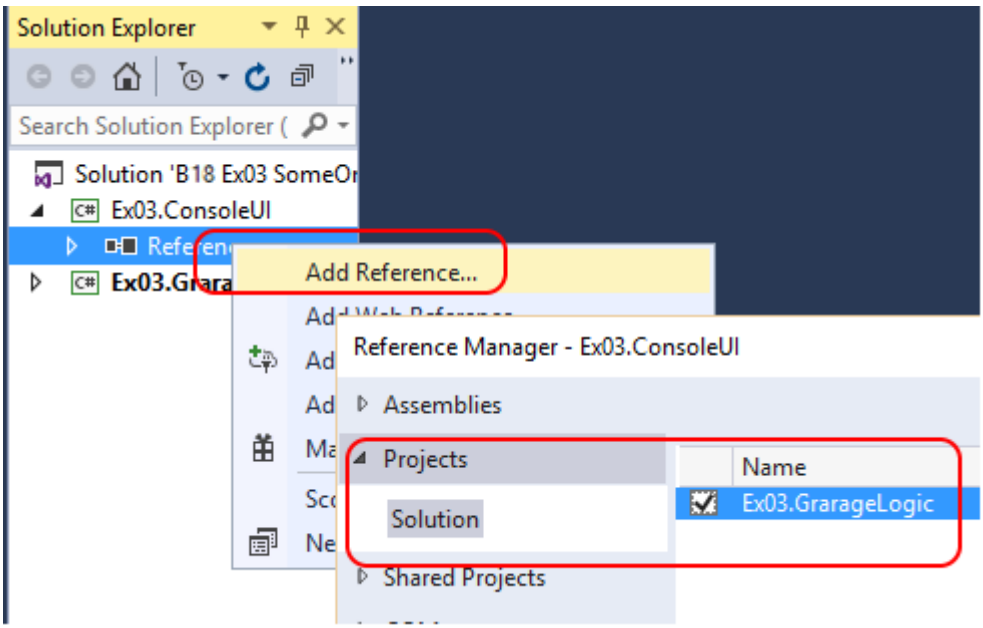
   To create a .dll file, acess the project's properties and choose **Output Type: Class Library**



2. **Ex03.ConsoleUI**

   This project creates the exe and incorporates the user interface implementation of the garage management system for the console.

   This system uses the model implemented in the first project by referencing it:



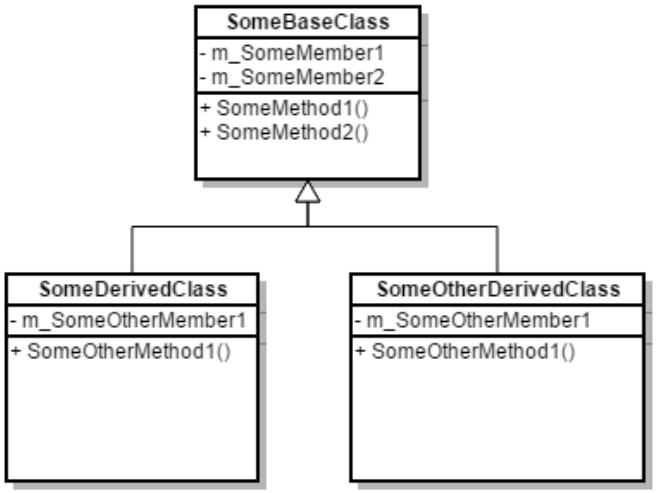
Remarks:

1. Design your system in the best manner of code, regarding dividing to classes and methods, inheritance and polymorphism.
2. Separate between the user interface and the logical layer which manages the object model and the logical entities of the system (A class like Vehicle should not know the Console class, not even indirectly) Try and think of solutions for this separation.
3. **Place the code which creates new object from the Vehicle class, and only it, in a separate class, in the logical part of the system. This class may also hold the supported vehicle types list. This class cannot address the user, neither directly, nor indirectly.**
4. Design and implement the software in such way in which adding a new vehicle type (i.e. Tractor) in the future will be possible **without changing any code** (except minor additions in the class from (3) that creates the vehicle objects)
5. Avoid code/logic duplication.
6. You **must** use:
    - List<T> **and/or** Dictionary<K,V>
    - Enum
    - String formatting
    - FormatException (thrown when an input is not correct regarding Parsing)
    - ArgumentException (thrown when an input is not correct logically – for example refueling with the wrong fuel type
    - ValueOutOfRangeException - a class you will write – (thrown when an input exceeds the allowed range, for example refueling with too much fuel) This class inherits from the Exception class, and holds additional fields:
        - float MaxValue
        - float MinValue

Submission:

Submit by Tuesday, May 15th, at 22:00. Every day late will result in a 2 point deduction. Furthermore, in addition to the text file containing the submission details in the solution, add a .docx file containing:

- Names and IDs of the submitters
- A list of types defined in your solution (class/enum) with a short explanation about each one.
- A diagram that describes the hierarchies of inheritance and relations between your classes. For example:



The file name will be the same as the solution name (with .docx suffix)

Good Luck!