C# Exercise 5 - Garage 1.0

NOTE - The result of the exercise must be shown to teachers and approved before it can be considered completed.

A first comprehensive project

To connect much of what you have learned, we will now build a garage application.
This application shall provide the functionality that a system may need if it is to be used to simulate a simple garage. It should therefore be possible to park vehicles, retrieve vehicles, check which vehicles are there and what characteristics they have. All this in a console application with main menu and submenus.

The reason why you should program a garage is that it is easy to anchor the division in the whole. We can mainly divide a garage into the following parts:

The garage: A representation of the building itself. The garage is a place where a variety of vehicles can be stored. The garage can thus be represented as a collection of vehicles.

Vehicles: Cars, motorcycles, unicycles or whatever type of vehicle you want to put in the garage.

These are the two "object types" that you see in a physical garage. But if we look more closely, there should also be subclasses for vehicles, meaning that each vehicle type is its own subclass in the system. In addition to this, it requires functionality that manages that vehicles are placed in the garage, that vehicles can be taken out of the garage, and that we can get a presentation of what is in the garage and search it.

In more programming-friendly terms, we should have as a minimum:

- A collection of vehicles; the Garage class.
- A vehicle class, the class Vehicle
- A number of subclasses for vehicles.
- A user interface that allows us to use the functionality of the garage. All interaction with the user takes place here.
- A GarageHandler. To abstract a layer so that there is no direct contact between the user interface and the garage class. This is conveniently done through a class that handles the functionality that the interface needs to have access to.

- We do not program directly against concrete types, so we use Interfaces for that, eg IUI, IHandler, IVehicle. (Tip is to break out to interface when the implementation is complete if you think this part is difficult)

Requirements specification

The vehicles must be implemented as the **Vehicle** class and its subclasses.

- **Vehicle** contains all properties that must be present in all vehicle types.
  For example. registration number, color, number of wheels and other characteristics you can think of. • The registration number is unique • There must be at least the following subclasses: •
  Airplane • **Motorcycle**
  • **Car**
  • **Bus**
  • **Boat**
- These must implement at least one own property each, e.g.:
  • ***Number of Engines*** •
  Cylinder volume •
  Fueltype (Gasoline/ Diesel) •
  Number of seats •
  Lenght

The garage itself should be implemented as a generic collection of vehicles:

```
class Garage<T>
```

In addition, the generic type must be restricted using a constraint:

```
class Garage<T> where ....
```

Furthermore, it should be possible to iterate over an instance of Garage using foreach. This means that Garage must implement the generic variant of the IEnumerable interface:

```
class Garage<T> : ....
```

The class does not need to inherit from any other class or implement any other interface.

The collection of vehicles must be handled internally in the class as an array, i.e. **Vehicle[].** The internal array should be **private.** When instantiating a new garage, the capacity must be specified as an argument to the constructor.

We should **NOT** use a **List<Vehicle>** internally in the Garage class!!!!

Functionality

It should be possible to: • List all parked vehicles • List vehicle types and how many of each are in the garage • Add and remove vehicles from the garage • Set a capacity (number of parking spaces) when instantiating a new garage • Possibility to populate the garage with a number of vehicles from the start. • Find a specific vehicle by registration number. It should work with both ABC123 and Abc123 or AbC123. • Search for vehicles based on one or more characteristics (all possible combinations from the Vehicle base class). For example:

- All black vehicles with four wheels.
- All motorcycles that are pink and have 3 wheels.
- All trucks • All red vehicles • The user should receive feedback that things went well or badly. For example, when we parked a vehicle, we want a confirmation that the vehicle is parked. If it doesn't work, the user wants to know why.

The program must be a console application with a text-based user interface.

From the interface it should be possible to:

• Navigate to all functionality from the garage via the interface • Create a garage with a user-specified size • It must be possible to shut down the application from the interface

The application must handle input data in a robust manner, so that it does not crash in the event of incorrect input or use.

Unit testing

The test must be created in a separate test project. We limit ourselves to testing the public methods of the Garage class. (Writing tests for the entire application is considered an extra task if there is time)

Feel free to experiment with writing the tests before you implement the functionality!
Then use ctrl. to generate your objects and methods.
Then implement the functionality until the test passes.

Structure the tests according to the principle.

1. **Arrange** here you set up what is to be tested, instantiate objects and inputs 2. **Act** here you call the method to be tested 3. **Assert** here you check that you got the expected result

Also remember to name the tests in an explanatory way. When a test fails, we want to know what didn't work just by looking at the test method name.

For example:
[MethodName_StateUnderTest_ExpectedBehavior]

public void Sum_NegativeNumberAs1stParam_ExceptionThrown()

BONUS: For those of you who still have time left

It is possible via C# to write and read to the file system from your application. Find out how to save your garage (via menu or automatically when shutting down) and load your garage (via menu or automatically when starting the application)

Option to also be able to search for the vehicle-specific characteristics.

Enter the size of the garage via configuration.

Manage multiple garages.

Any functionality you think should be available.

Good luck!