Second Checkpoint

In the previous checkpoint we created a procedural application. Everything in one file, without any object orientation concepts applied. Now that you are familiar with some object orientation principles go back and look at your checkpoint01 solution. You will probably see several places where improvements can be done.

This is exactly what we are going to do, improve and extend the previous checkpoint. From here onwards, each checkpoint will get closer and closer to real world scenarios.

Description

In this exercise you will add more functionality to the previous software, instead of just listing, you will create a small vehicles CRUD (create, read, update and delete)

To make things simple, let's split this exercise in two parts, refactoring and new content.

Let's first start with refactoring by applying object orientation concepts to the previous checkpoint, we recommend you create a new project and copy the content from the previous checkpoint, this way you can check your progress.

1. Create a class that will represent Automaker
    1. With an attribute called name.
    2. An Automaker object can only be created if name is passed
2. Create a class that will represent a vehicle.
3. Vehicle has an association with Automaker, each Vehicle must have one Automaker, update the class Vehicle to represent this change, also:
    1. Add two more attributes
        1. color
        2. year
    2. Make sure the Vehicle object is always created with all attributes
4. Pay attention to the access modifiers for the attributes and the correct methods to get and set values to them.
5. Create a method called prettyPrint in the Vehicle class to print the object attributes data
6. Create a new class that will represent our repository (database), you can call it VehicleRepository
    1. Create an attribute to hold an array of vehicles. This array of vehicles is the place where we will apply our CRUD operations.
    2. Encapsulate and move the creation of the previous multidimensional vehicles array in the Application class into a method in the VehicleRepository class.
    3. Since we are working with Objects there is no need to have multidimensional arrays, since Vehicle object has the model and automaker attributes, we just need a unidimensional array with Vehicle objects, update your code to reflect that.
    4. Create a constructor inside VehicleRepository that calls the method initializing the Vehicle array and assign it to the class attribute
    5. Create the proper get and set
7. Create a new class to hold the business logic, you can call it VehicleService for example, and inside this class:
    1. Create a VehicleRepository attribute and initialize it. Question can this attribute be final?
    2. Create a method called, searchByAutomaker,a String representing the Automaker name and returns a new array with that Automaker vehicles
    3. In the Application class, update the menu to add the searchByAutomaker and when triggered you should call the searchByAutomaker from the VehicleService, receive the array of vehicles back and print its data. The menu should be similar to this:

```
1. - Search by automaker
0 - Exit
```

--

At this moment we finished refactoring what we have done before. Time to add more features to our software:

1. Inside the VehicleService create another method called `searchByModel`, a String representing the model name and returns a Vehicle object
2. In the Application class, update the menu to include the `2. Search by model` and print the vehicle when the data is returned.
3. Inside the VehicleService create another method called `addVehicle`, this method should receive a Vehicle object and add it to the existing array of Vehicles in VehicleRepository (Resize if necessary)
4. In the Application class, update the menu to include the `3. Add vehicle`, ask the properties for the user, create the object and pass to addVehicle. Use the list by model/automaker to validate if it was inserted
5. Inside the VehicleService create another method called `updateVehicle`, this method should receive, two Vehicle objects, the old and the new. Update the array to replace the old object with the new one
6. In the Application class, update the menu to include the `4. Update vehicle`, for now let's assume the vehicles models are unique, find the correct Vehicle to be updated by its model and pass it to the updateVehicle alongside the new Vehicle object. Use the search to find the updated vehicle.
7. Inside the VehicleService create another method called `deleteVehicleByModel`, this method should receive the model to be used to search for the vehicle to be deleted. If the vehicle is present in the array, remove it and make sure the array does not have an empty value in the middle of it.
8. In the Application class, update the menu to include the `5. Delete vehicle`, ask the model for the user, pass to `deleteVehicleByModel` and print that the Vehicle was deleted successfully. Use the search to check if it was deleted successfully

Note: Even though this checkpoint is very detailed, its purpose is to give you some direction, feel free to use your own style of programming, create more classes, methods etc. The most important thing is to fulfill the requirements and add the proper checks for the bad scenarios