<u>The assignment:</u>

Please use knowledge of all ***chapters 1 - 10 from this course****. Also, all the knowledge from the prerequisite courses.*

This is your final. In this assignment, you will create an application that sells a product like a Car Dealership application. It can be any other product of your choice. You are free to use your creativity and play around with the concepts you have learned in this course and the courses before this class. You can review old courses' lecture videos under Review section for OOP fundamentals and about designing your SQL database.

Upload:

- A **zip** file that includes your code to Canvas.
- A 10-minute recorded video (**mp4**) to the Discussions on Canvas that includes:
    - Design of the code and how you came up with it
    - A walkthrough of the code pointing out specifically how things were done
    - Unit tests and demo of the tests
    - A demo of the application
    - Showcase all the validations in your code as well as when you demo the application
    - **The whole group should participate in the video**

The front end for the Car Dealership will have:

- List of cars for sale even if the user is not logged in. This list should display cars

  as:

    - The car listings should be sorted by it's creation date and time with newer listings on top. A **View All** option should bring you back to this view.
    - When a listing is clicked, show the details of the listing. This can include the user that uploaded the listing, pictures of the car, and any other properties from your design.
    - A **Filter By** option. This should display filter options like filter by make, model, color, age, and price.
    - When a filter option is selected, the application should show subsections with **titles** in ascending order.
        - The titles can include the different makes in ascending order if make is selected. Then display appropriate car listings underneath each title ordered by listings' creation times.
        - There is no need to filter by model so you can skip that.

        - ✦ If color is selected, figure out how many different colors are in your saved car listings data and create that many subsections with titles as those colors in ascending order. Then display the cars under the appropriate color ordered by listings' creation times.
        - ✦ The age and price should have ranges. For example: price under $5000, between $5000 to $9,999, and $10,000+ as the titles for the subsections. Then display the appropriate car listings in those subsections ordered by listings' creation times.
    - **Upload** to upload a car listing. Make sure the user exists and is logged in before creating a car listing.
    - **Delete** should delete the car listing from the application if the user is logged in.
    - **Comment** section where a potential buyer can ask questions to the seller.

- User:
    - Register a user and ask them to provide login username and password. Make sure the user doesn't already exist. Can you save the password in encrypted format in your database? Use your creativity here.
    - Think about the roles and what each role is allowed to see or do.
    - Use the login and password to open up the seller's view where they can see all their listings. Again, this will be part of your design. How many forms do you want to include and what will they all look like?
    - Include functionality for a user who wants to delete their account.
- **Pagination** to show only a set of records per page.
- **Exit** to exit the application(s).

The backend:

- Design and create your SQL database. Connect it to your application using **DataSource** and use the **DataSet** classes.

The code:

Please use OOP fundamentals and everything else you have learned in this class and the classes before. Following is a guideline on how you can design your system. You are free to change it and make it your own:

- Create an abstract class called Car. Then create classes that inherit from it.
    - The Car class should have all the common properties: make, model, color, age, and price.
    - The subclasses can be any car company name, which is the make and can also be used as the value for the make property. For example: BMW, Toyota, Honda,

Mercedes, etc. are the 4 subclasses and their make properties are also the string names BMW, Toyota, Honda, and Mercedes respectively. You can use the .GetType() method to get the string names for the Make properties.

  - In addition, the subclasses should have one or more other-features as properties. For example: Engine for BMW, Mileage for Toyota, etc. to highlight that company's best feature(s).
  - For each subclass, override the .ToString() method to display all the properties in some order along with the highlight feature(s) included. This method can be used to display the text for the car listing on the main form. However, if you are displaying text after using the Filter By option, make sure you don't include the filtered property in the text. For example: If the user is filtering by color, make sure to skip the "color: Black" property from the text returned from the .ToString() method. However, if the filter is age or price, then that **should** be included. You are free to do this any way you want. Maybe create overloaded methods?
- A Listing class should have the properties for Car and the DateTime of the creation of the listing. This DateTime is the date and time of when the Upload button is clicked and the user enters the necessary information and clicks on Accept.
- Change the code so you use Interfaces instead.
  - Create your own custom interfaces.
  - Use the in-built interfaces that help you compare different listings while filtering. *(Hint: IComparable)*
- Use Generics <*T*> for the different cars' classes.
- Make sure to not have all the code in one class with a lot of if-else statements. Instead have specific code inside of each of the individual cars classes.
- Use Indexers, Events, Delegates, Operator, and/or LINQ in your code.
- For pagination, maybe you want to write a SQL query that gets you the records in chunks and you load your form with those records per page.
- You are free to pick your own data structure(s) for this assignment. I recommend using a list or a sorted list.
- Think about what kinds of validations you need for your form and your backend code and implement those.
- Unit tests are also required. Please use your knowledge from 166 to write unit tests classes with the [TestClass] and [TestMethod] tags.
- Think about the **3-tiered architecture** when designing your application. Split your classes in those three layers.