**Building an Automobile Management Application with Windows Forms**

# Introduction

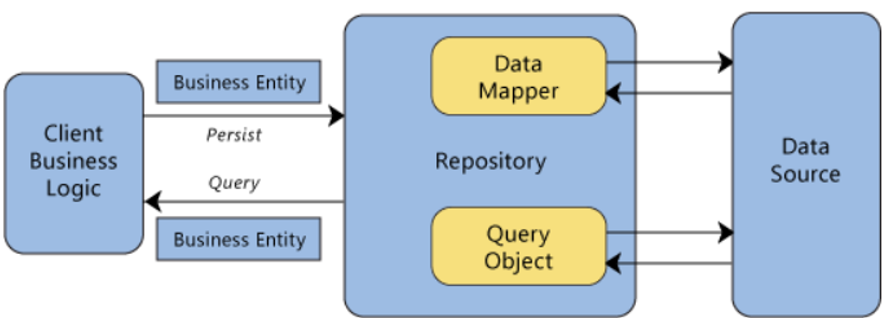
Imagine you're an employee of a car retailer named **Automobile Store**. Your manager has asked you to develop a Windows Forms application for automobile management (CarID, CarName, Manufacturer, Price, and ReleasedYear). The application has to support adding, viewing, modifying, and removing products—a standardized usage action verbs better known as Create, Read, Update, Delete (CRUD).

This lab explores creating an application using Windows Forms with .NET Core, and C#. An "in-memory database" will be created to persist the car's data, so a collection is called **List** will be used for reading and managing automobile data.

This lab is applying Repository and Singleton Pattern. A Repository in C# mediates between the domain and data mapping layers. It allows you to pull a record or number of records out of datasets, and then have those records to work on acting like an in-memory domain object collection, and you can also update or delete records within those data set, and the mapping code encapsulated by the Repository will carry out the appropriate operations behind the scenes.

Repository pattern C# is a way to implement data access by encapsulating the set of objects persisted in a data store and the operations performed over them, providing a more object-oriented view of the persistence layer.

Repository pattern C# also supports the objective of achieving a clean separation and one-way dependency between the domain and data mapping layers.



# Lab Objectives

In this lab, you will:

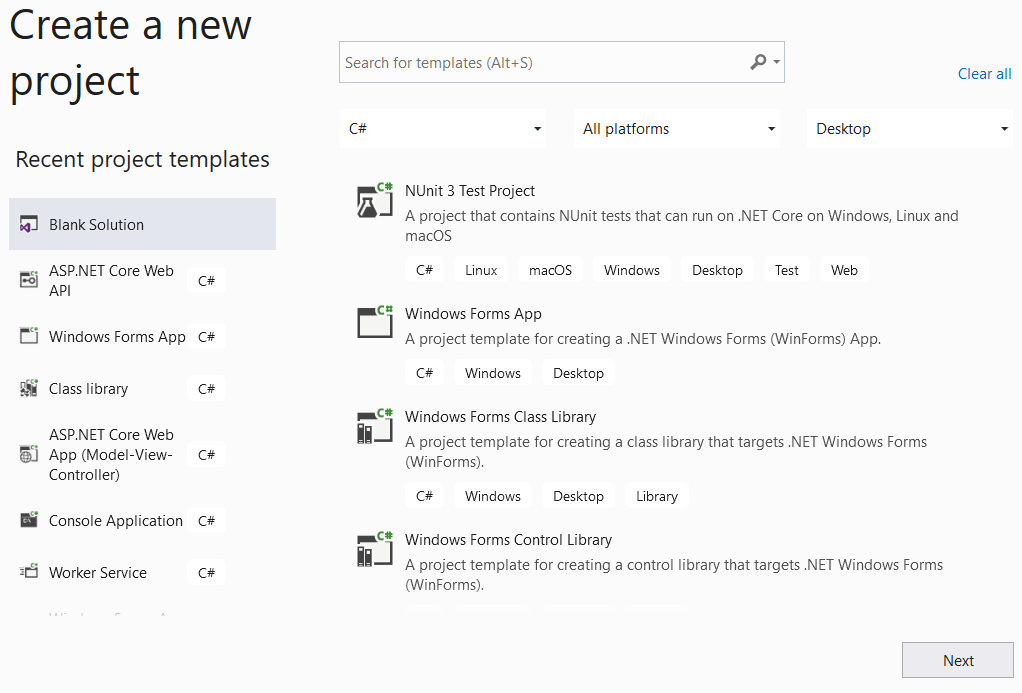
* Use the Visual Studio.NET to create Windows Forms and Class Library (.dll) project.
* Create a List of persisting cars using LinQ to Object to find cars.
* Apply passing data in WinForms application
* Apply Repository pattern and Singleton pattern in a project
* Add CRUD action methods to WinForms application.
* Run the project and test the WinForms actions.

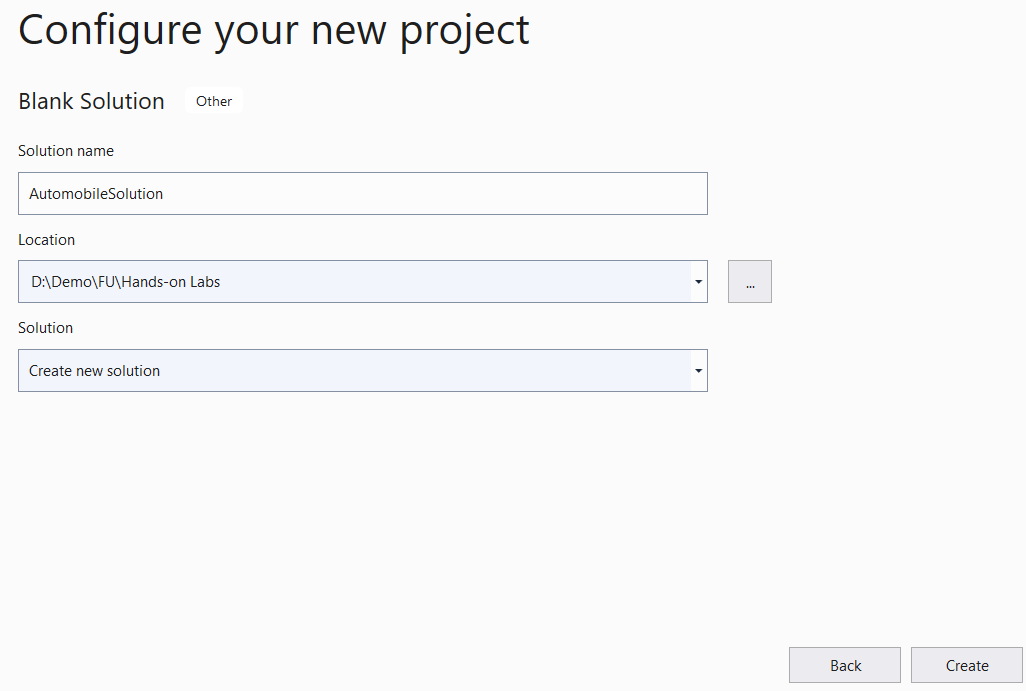
# Activity 01: Build a solution by Visual Studio.NET

Create a Blank Solution named **AutomobileSolution** then add new a Class Library Project named **AutomobileLibrary** and a Windows Forms project named **AutomobileWinApp**

**Step 01**. Create a Blank solution.

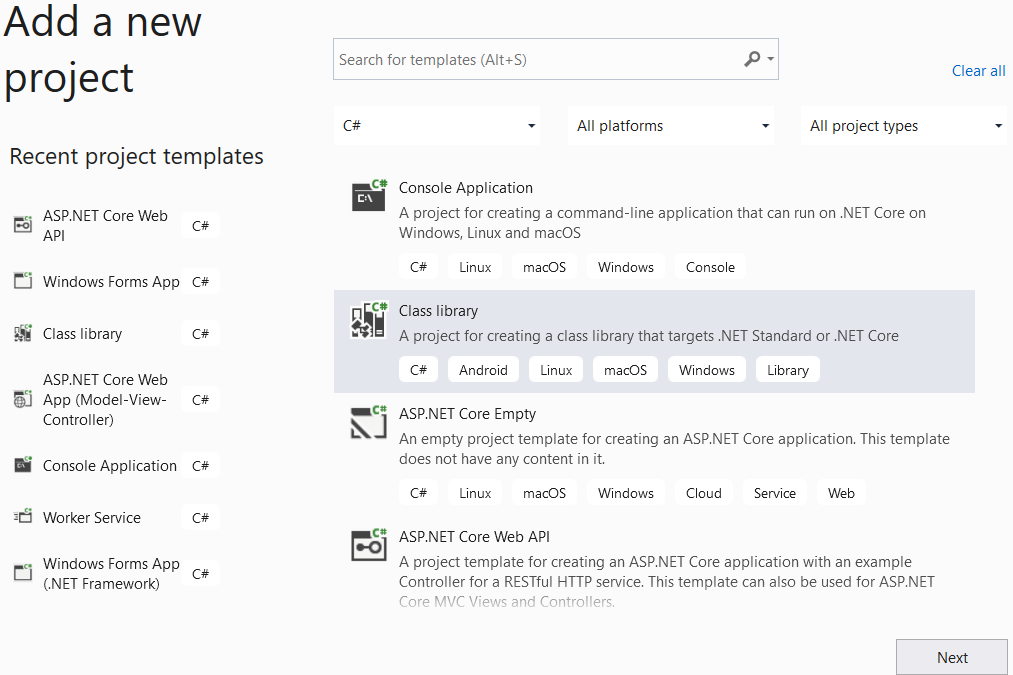
* Open the Visual Studio .NET application and performs steps as follows:

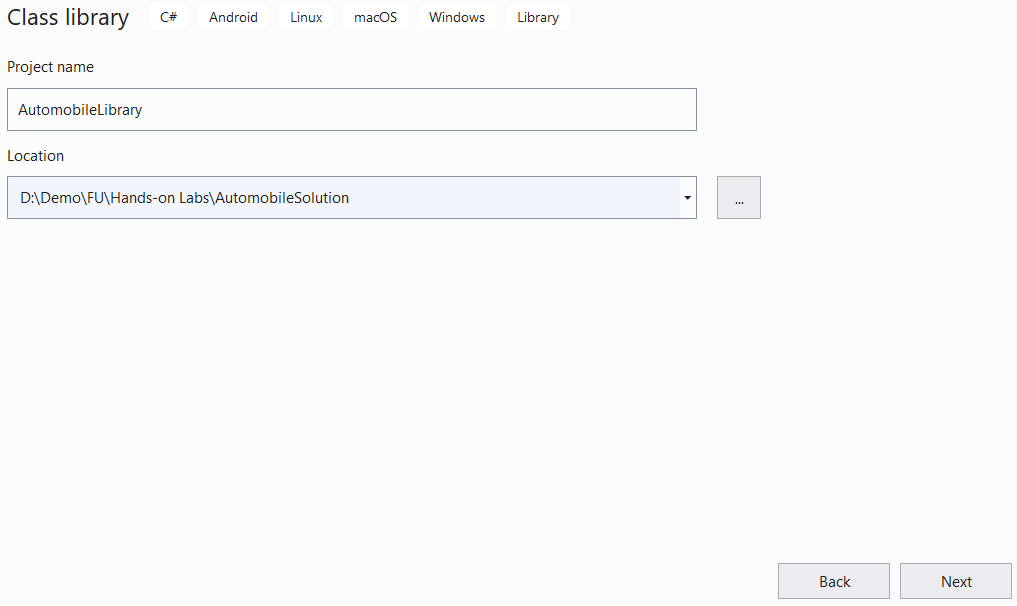




**Step 02.** Create a Class Library project.

* From the File menu | Add | New Project, on the Add New Project dialog, select “Class Library” and performs steps as follows:



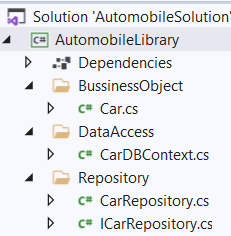


****

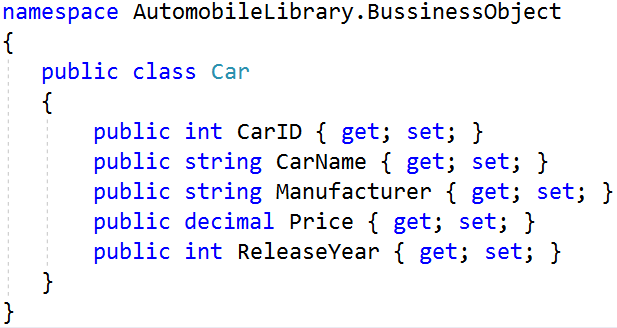
**Step 03.** Repeat **Step 02** to create a Windows Form project.

# Activity 02: Write codes for the AutomobileLibrary project

**Step 01**. Create folders and add class to the project as follows:

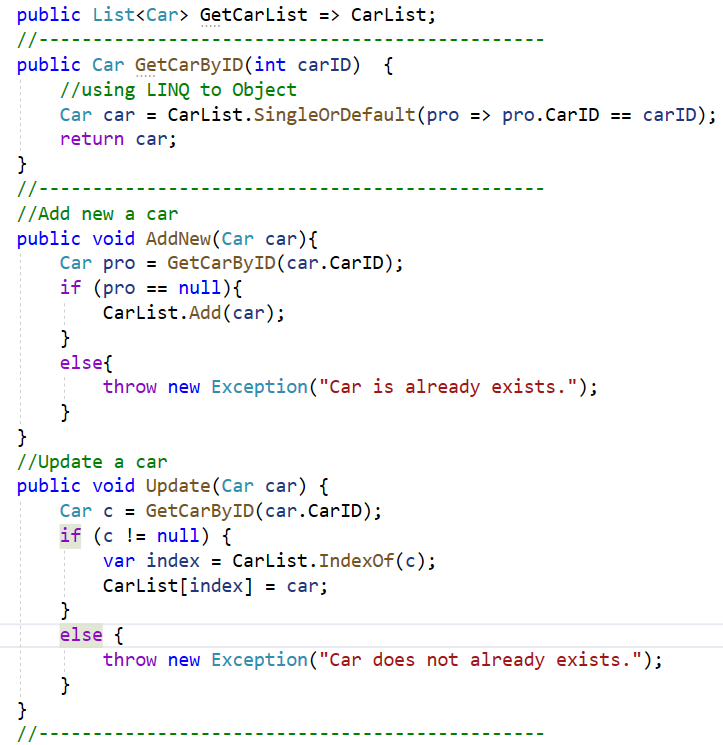


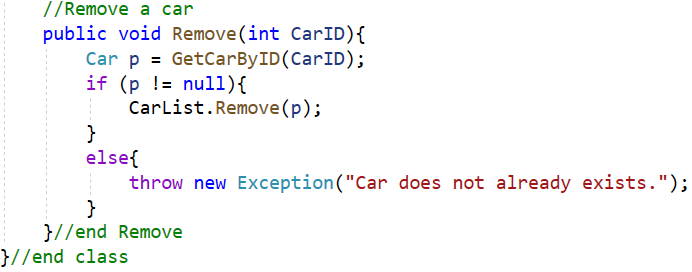
**Step 02**. Write code for **Car.cs** as follows:



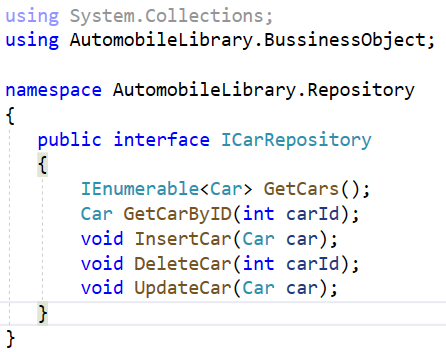
**Step 03**. Write code for **CarDBContext.cs** as follows:



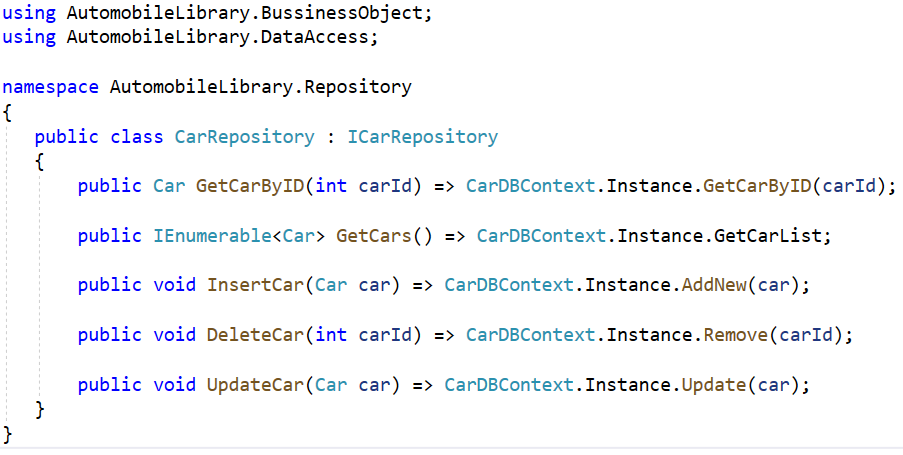




**Step 04**. Write codes for **ICarRepository.cs** as follows:



**Step 05**. Write codes for **CarRepository.cs** as follows:

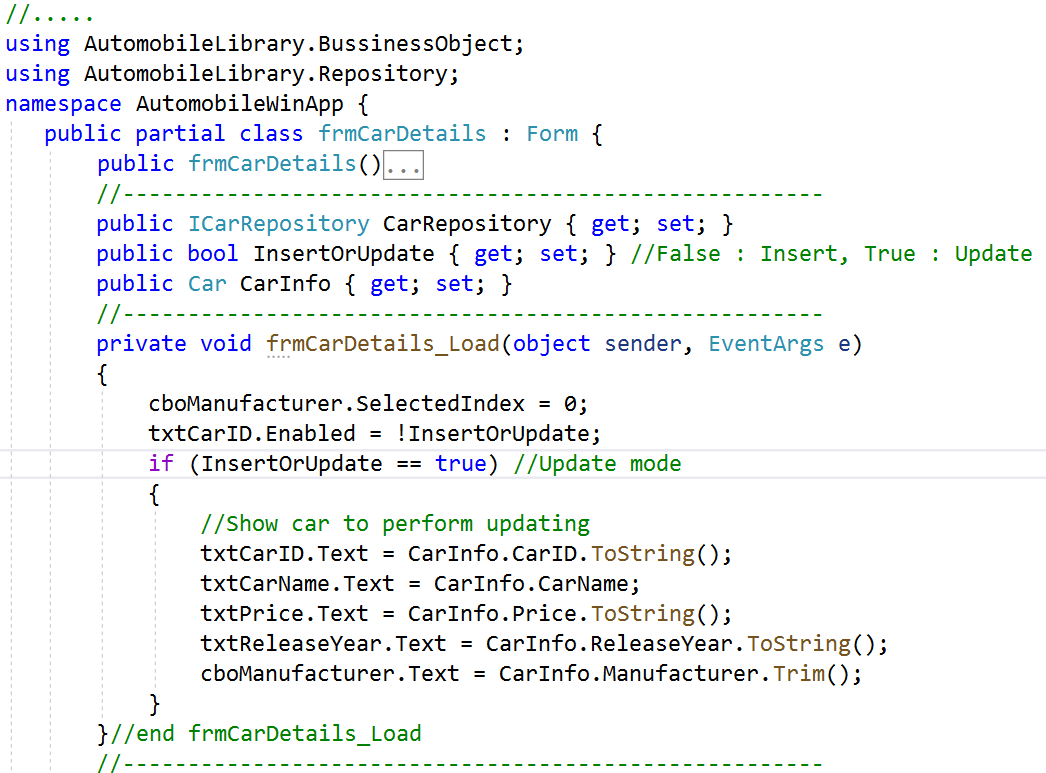


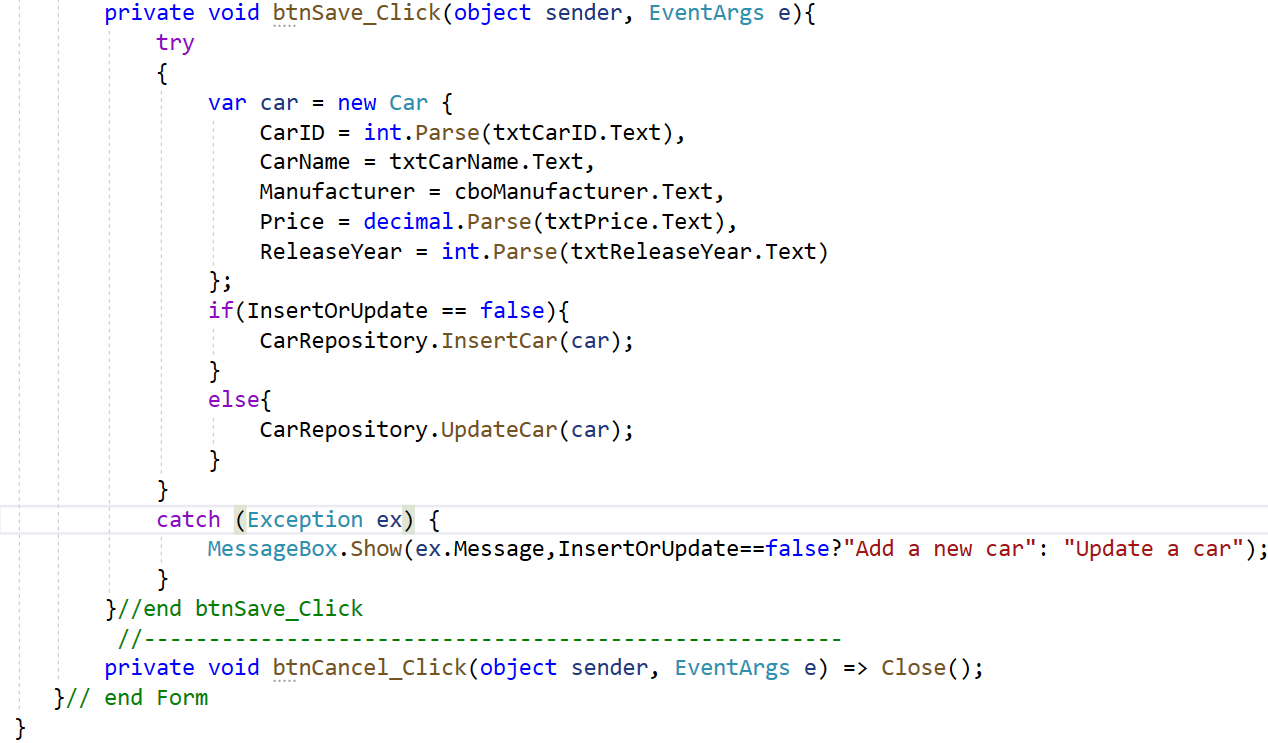
# Activity 03: Design UI and write codes for WinForms project

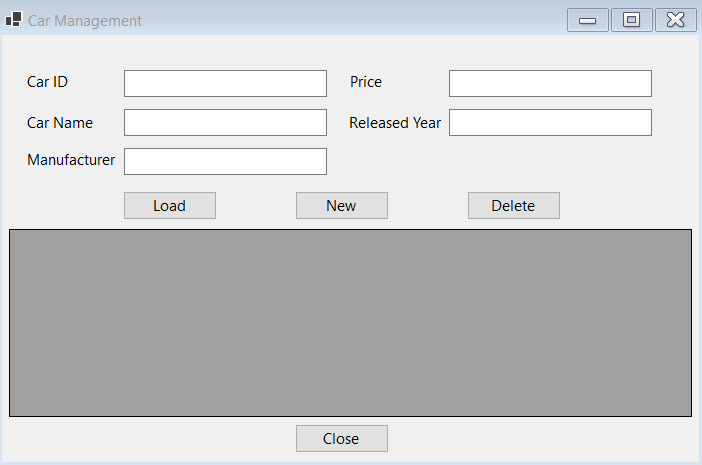
**Step 01**. Right-click on the **AutomobileWinApp** project and add a new form named **frmCarDetails.cs** with UIas follows:

# 

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Object Type | Object name | Properties / Events |
| 1 | Label | lbCarID | Text: Car ID |
| 2 | Label | lbCarName | Text: Car Name |
| 3 | Label | lbManufacturer | Text: Manufacturer |
| 4 | Label | lbPrice | Text: Price |
| 5 | Label | lbReleaseYear | Text: ReleaseYear |
| 6 | TextBox | txtCarID |  |
| 7 | TextBox | txtCarName |  |
| 8 | MaskedTextBox | txtPrice | Mask: 000000000  Text: 0 |
| 9 | MaskedTextBox | txtReleaseYear | Mask: 0000  Text: 0 |
| 10 | ComboBox | cboManufacturer | Items:  Audi  BMW  Ford  Honda  Hyundai  Kia  Suzuki  Toyota |
| 11 | Button | btnSave | Text: Save  DialogResult: OK  Event Handler: Click |
| 12 | Button | btnCancel | Text: Cancel  DialogResult: Cancel  Event Handler: Click |
| 13 | Form | frmCarDetails | StartPosition: CenterScreen  Text: frmCarDetails  Event Handler: Load |

**Step 02**. Write codes for **frmCarDetails.cs**:

**Step 03**. Design UI for **frmCarManagement.cs** form:



|  |  |  |  |
| --- | --- | --- | --- |
| No. | Object Type | Object name | Properties / Events |
| 1 | Label | lbCarID | Text: Car ID |
| 2 | Label | lbCarName | Text: Car Name |
| 3 | Label | lbManufacturer | Text: Manufacturer |
| 4 | Label | lbPrice | Text: Price |
| 5 | Label | lbReleaseYear | Text: ReleaseYear |
| 6 | TextBox | txtCarID |  |
| 7 | TextBox | txtCarName |  |
| 8 | TextBox | txtPrice |  |
| 9 | TextBox | txtReleaseYear |  |
| 10 | TextBox | txtManufacturer |  |
| 11 | Button | btnLoad | Text: Load  Event Handler: Click |
| 12 | Button | btnNew | Text: New  Event Handler: Click |
| 13 | Button | btnDelete | Text: Delete  Event Handler: Click |
| 14 | DataGridView | dgvCarList | ReadOnly: True  SelectionMode:FullRowSelect |
| 15 | Form | frmCarManagement | StartPosition: CenterScreen  Text: Car Management  Event Handler: Load |

# Activity 04: Reference to AutomobileLibrary project and write code for WinForms project

# Step 01. Right-click on AutomobileWinApp project, select Add | Project Reference, and perform as the below figure:

# 

# Step 02. Write codes for frmCarManagement.cs

# 

# 

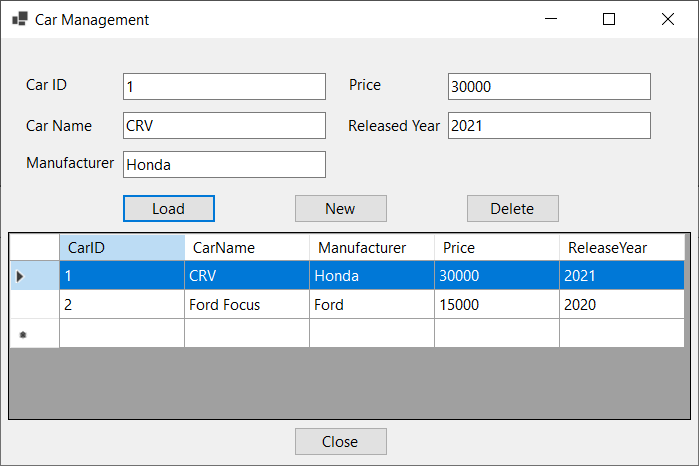
# 

# 

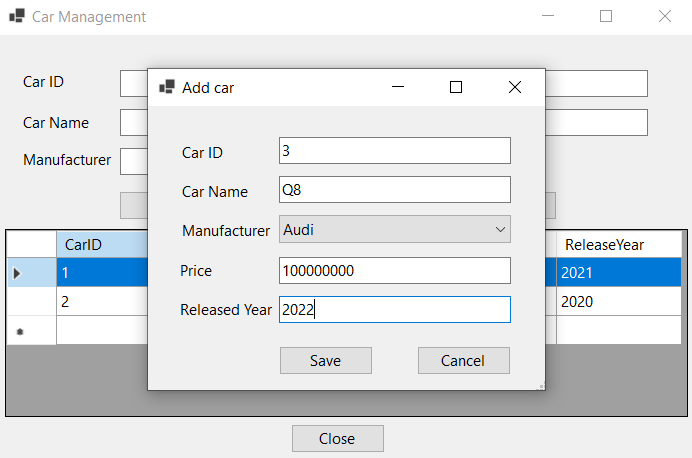
# 

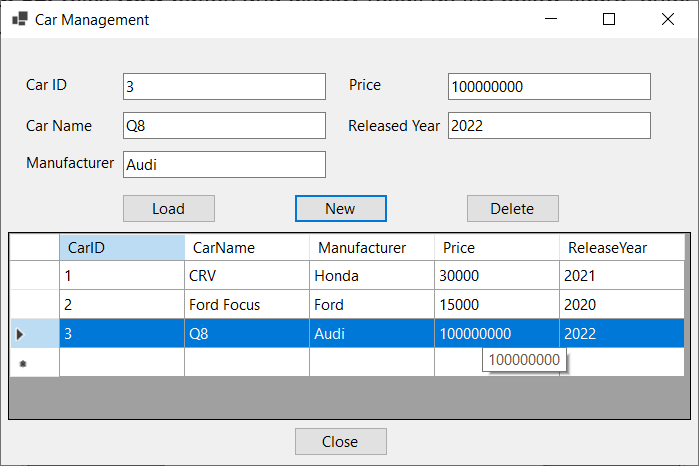
# Activity 06: Press Ctrl+F5 to run the WinForms project and test all actions

# Step 01. Click Load button and display the result as the below figure.

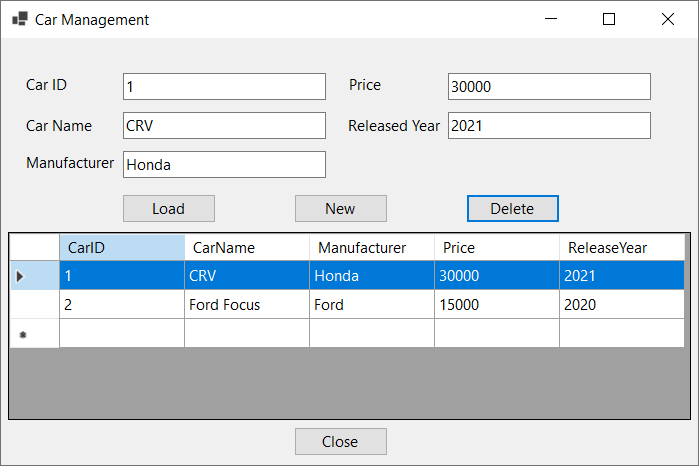


# Step 02. Click New button and display the result as the below figure, enter the values on TextBoxes then click Save





# Step 03. Select a row on the DataGridView then click Delete to remove a Car



# Step 04. Double-click a row on the DataGridView to update a Car on the popup form, edit values then click Save

# 

# 