

Building an Automobile Management Application with Windows Forms

Introduction

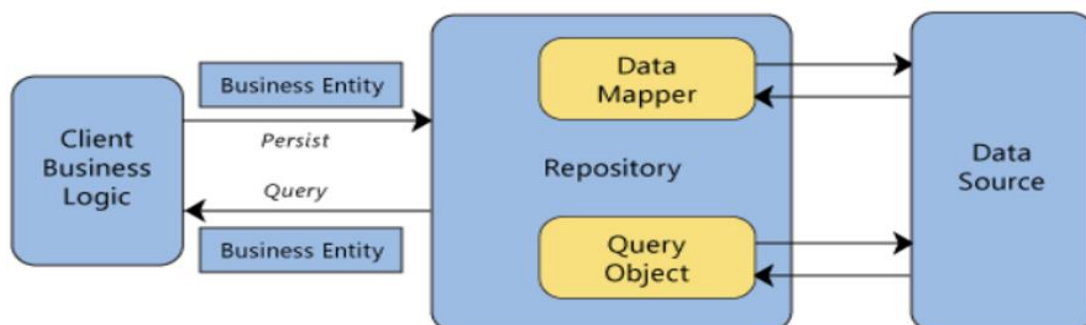
Imagine you're an employee of a car retailer named **Automobile Store**. Your manager has asked you to develop a Windows Forms application for automobile management (CarID, CarName, Manufacturer, Price, and ReleasedYear). The application has to support adding, viewing, modifying, and removing products—a 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:

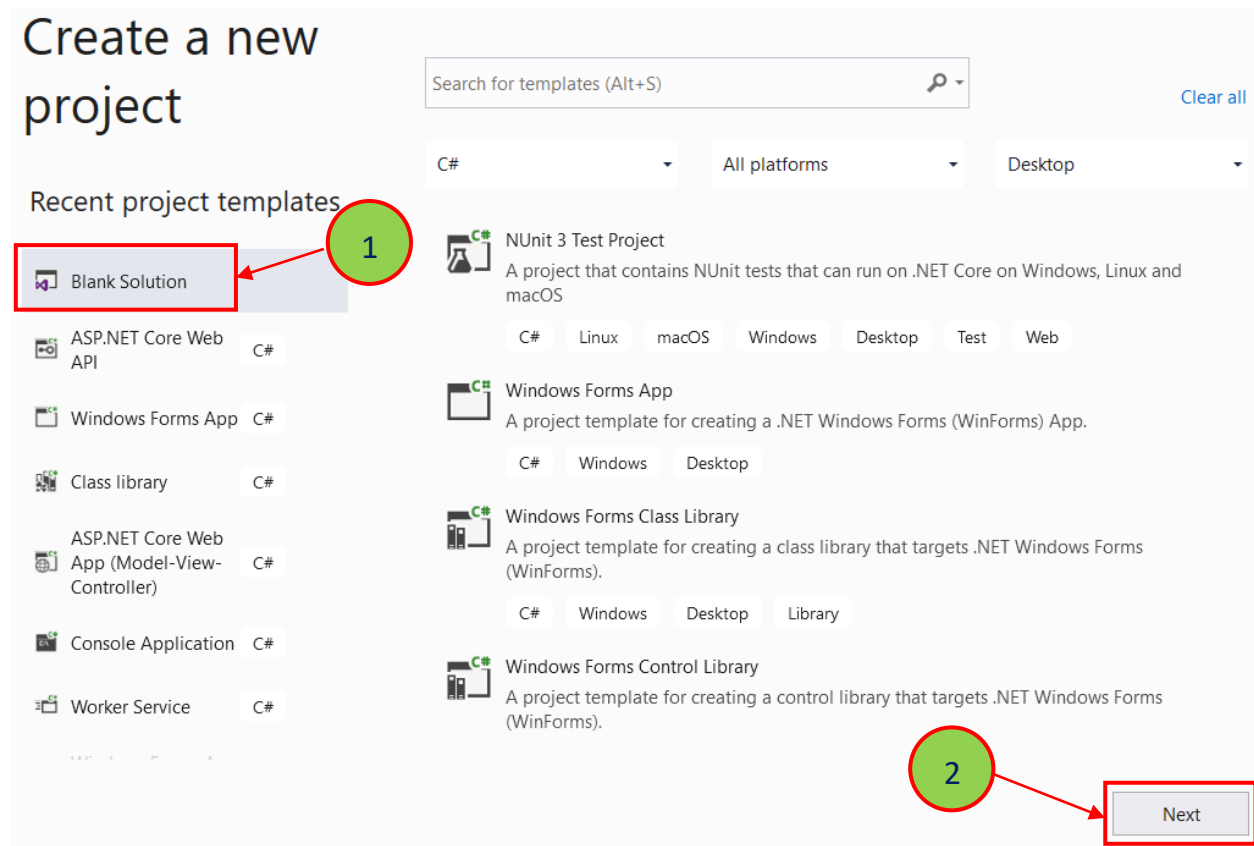
- Use the Visual Studio.NET to create Windows Forms and Class Library (.dll) project.
- Create a List of persisting cars using LinQ to Object to find cars.
- Apply passing data in WinForms application
- Apply Repository pattern and Singleton pattern in a project
- Add CRUD action methods to WinForms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



The screenshot shows the 'Create a new project' window in Visual Studio. The 'Recent project templates' list on the left includes 'Blank Solution', 'ASP.NET Core Web API', 'Windows Forms App', 'Class library', 'ASP.NET Core Web App (Model-View-Controller)', 'Console Application', and 'Worker Service'. The 'Blank Solution' template is highlighted with a red box, and a green circle with the number 1 points to it. The 'Search for templates (Alt+S)' search bar is at the top. The 'C#' language, 'All platforms', and 'Desktop' target are selected. The 'Next' button at the bottom right is highlighted with a red box, and a green circle with the number 2 points to it.

Configure your new project

Blank Solution

Solution name 3

AutomobileSolution 4

Location

D:\Demo\FU\Hands-on Labs

Solution

Create new solution

Back Create 5

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:

Add a new project

Recent project templates

- ASP.NET Core Web API C#
- Windows Forms App C#
- Class library C#
- ASP.NET Core Web App (Model-View-Controller) C#
- Console Application C#
- Worker Service C#
- Windows Forms App (.NET Framework) C#

Search for templates (Alt+S)

C# All platforms All project types

Console Application
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
C# Linux macOS Windows Console

Class library
A project for creating a class library that targets .NET Standard or .NET Core
C# Android Linux macOS Windows Library 1

ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Next 2

Class library C# Android Linux **3** Windows Library

Project name

AutomobileLibrary

Location

D:\Demo\FU\Hands-on Labs\AutomobileSolution

Back **4** Next

Additional information

Class library C# Android Linux macOS **5** Windows Library

Target Framework

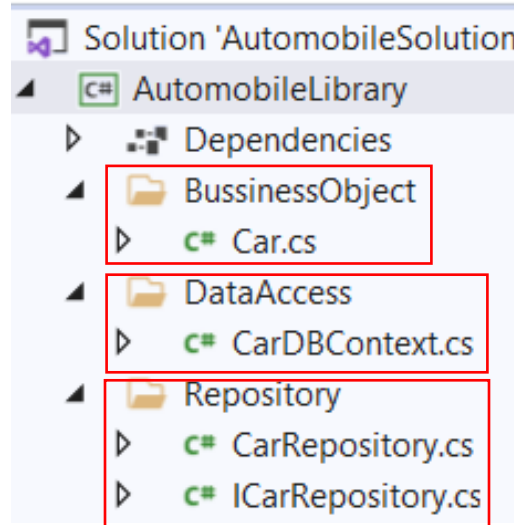
.NET 5.0 (Current)

Back **6** Create

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:

```
namespace AutomobileLibrary.BussinessObject
{
    public class Car
    {
        public int CarID { get; set; }
        public string CarName { get; set; }
        public string Manufacturer { get; set; }
        public decimal Price { get; set; }
        public int ReleaseYear { get; set; }
    }
}
```

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

```
using AutomobileLibrary.BussinessObject;
namespace AutomobileLibrary.DataAccess
{
    public class CarDBContext{
        //Initialize car list
        private static List<Car> CarList = new List<Car>(){
            new Car{ CarID=1, CarName="CRV", Manufacturer="Honda",
                Price=30000, ReleaseYear=2021 },
            new Car{ CarID=2, CarName="Ford Focus", Manufacturer="Ford",
                Price=15000, ReleaseYear=2020 }
        };
        //-----
        //Using Singleton Pattern
        private static CarDBContext instance = null;
        private static readonly object instanceLock = new object();
        private CarDBContext() { }
        public static CarDBContext Instance
        {
            get
            {
                lock (instanceLock)
                {
                    if (instance == null)
                    {
                        instance = new CarDBContext();
                    }
                    return instance;
                }
            }
        }
    }
    //-----
}
```

```

public List<Car> GetCarList => CarList;
//-----
public Car GetCarByID(int carID) {
    //using LINQ to Object
    Car car = CarList.SingleOrDefault(pro => pro.CarID == carID);
    return car;
}
//-----
//Add new a car
public void AddNew(Car car){
    Car pro = GetCarByID(car.CarID);
    if (pro == null){
        CarList.Add(car);
    }
    else{
        throw new Exception("Car is already exists.");
    }
}
//Update a car
public void Update(Car car) {
    Car c = GetCarByID(car.CarID);
    if (c != null) {
        var index = CarList.IndexOf(c);
        CarList[index] = car;
    }
    else {
        throw new Exception("Car does not already exists.");
    }
}
//-----
//Remove a car
public void Remove(int CarID){
    Car p = GetCarByID(CarID);
    if (p != null){
        CarList.Remove(p);
    }
    else{
        throw new Exception("Car does not already exists.");
    }
}
} //end Remove
} //end class

```

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

```
using System.Collections;
using AutomobileLibrary.BussinessObject;

namespace AutomobileLibrary.Repository
{
    public interface ICarRepository
    {
        IEnumerable<Car> GetCars();
        Car GetCarByID(int carId);
        void InsertCar(Car car);
        void DeleteCar(int carId);
        void UpdateCar(Car car);
    }
}
```

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

```
using AutomobileLibrary.BussinessObject;
using AutomobileLibrary.DataAccess;

namespace AutomobileLibrary.Repository
{
    public class CarRepository : ICarRepository
    {
        public Car GetCarByID(int carId) => CarDBContext.Instance.GetCarByID(carId);

        public IEnumerable<Car> GetCars() => CarDBContext.Instance.GetCarList;

        public void InsertCar(Car car) => CarDBContext.Instance.AddNew(car);

        public void DeleteCar(int carId) => CarDBContext.Instance.Remove(carId);

        public void UpdateCar(Car car) => CarDBContext.Instance.Update(car);
    }
}
```

Activity 03: Design UI and write codes for AutomobileWinApp project

Step 01. Rename **Form1.cs** to **frmCarManagement.cs**

Step 02. Right-click on the **AutomobileWinApp** project and add a new form named **frmCarDetails.cs** with UI as follows:

frmCarDetails

Car ID

Car Name

Manufacturer

Price

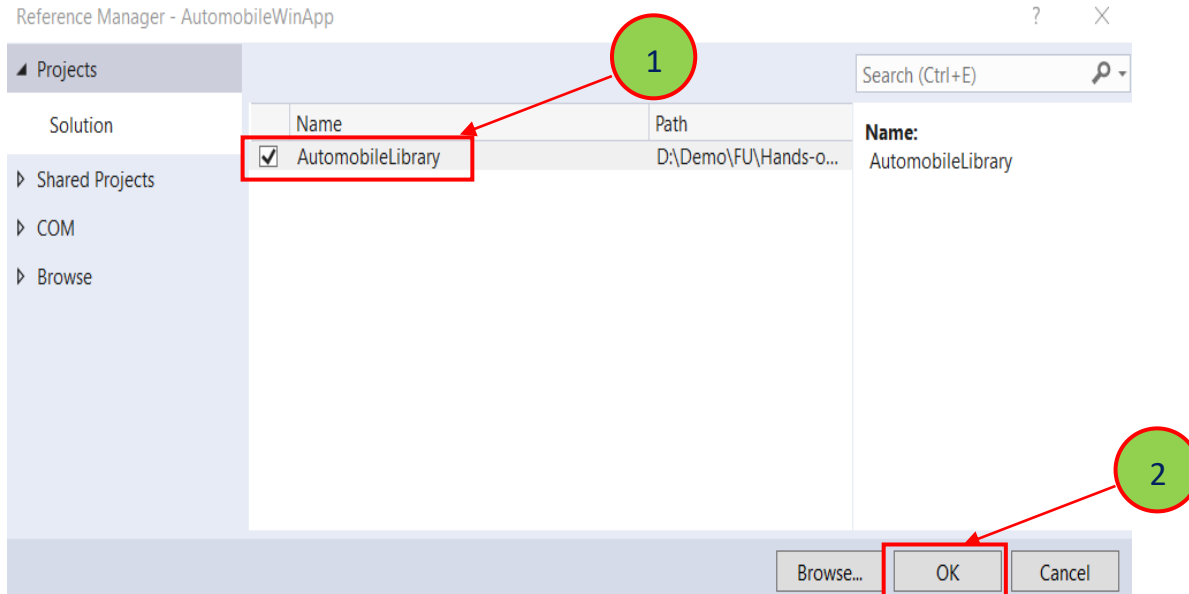
Released Year

Save Cancel

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 03. Reference to **AutomobileLibrary** project

- Right-click on **AutomobileWinApp** project, select Add | Project Reference, and perform as the below figure:



Step 04. Write codes for **frmCarDetails.cs**:

```
//.....
using AutomobileLibrary.BussinessObject;
using AutomobileLibrary.Repository;
namespace AutomobileWinApp {
    public partial class frmCarDetails : Form {
        public frmCarDetails() {
            //-----
            public ICarRepository CarRepository { get; set; }
            public bool InsertOrUpdate { get; set; } //False : Insert, True : Update
            public Car CarInfo { get; set; }
            //-----
            private void frmCarDetails_Load(object sender, EventArgs e)
            {
                cboManufacturer.SelectedIndex = 0;
                txtCarID.Enabled = !InsertOrUpdate;
                if (InsertOrUpdate == true) //Update mode
                {
                    //Show car to perform updating
                    txtCarID.Text = CarInfo.CarID.ToString();
                    txtCarName.Text = CarInfo.CarName;
                    txtPrice.Text = CarInfo.Price.ToString();
                    txtReleaseYear.Text = CarInfo.ReleaseYear.ToString();
                    cboManufacturer.Text = CarInfo.Manufacturer.Trim();
                }
            }
        }
    }
}
```

```
private void btnSave_Click(object sender, EventArgs e){
    try
    {
        var car = new Car {
            CarID = int.Parse(txtCarID.Text),
            CarName = txtCarName.Text,
            Manufacturer = cboManufacturer.Text,
            Price = decimal.Parse(txtPrice.Text),
            ReleaseYear = int.Parse(txtReleaseYear.Text)
        };
        if(InsertOrUpdate == false){
            CarRepository.InsertCar(car);
        }
        else{
            CarRepository.UpdateCar(car);
        }
    }
    catch (Exception ex) {
        MessageBox.Show(ex.Message, InsertOrUpdate==false?"Add a new car": "Update a car");
    }
} //end btnSave_Click
//-----
private void btnCancel_Click(object sender, EventArgs e) => Close();
} // end Form
}
```

Step 05. Design UI for frmCarManagement.cs form:

The screenshot shows a Windows application window titled "Car Management". The window contains the following elements:

- Text Fields:**
 - Car ID
 - Car Name
 - Manufacturer
 - Price
 - Released Year
- Buttons:**
 - Load
 - New
 - Delete
 - Close (at the bottom center)
- Display Area:** A large gray rectangular area below the buttons, intended for displaying a list of cars.

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

Step 06. Write codes for **frmCarManagement.cs**

```
using AutomobileLibrary.Repository;
using AutomobileLibrary.BussinessObject;
namespace AutomobileWinApp{
    public partial class frmCarManagement : Form{
        ICarRepository carRepository = new CarRepository();
        //Create a data source
        BindingSource source;
        //-----
        public frmCarManagement(){...}
        //-----
        private void frmCarManagement_Load(object sender, EventArgs e){
            btnDelete.Enabled = false;
            //Register this event to open the frmCarDetails form that performs updating
            dgvCarList.CellDoubleClick += DgvCarList_CellDoubleClick;
        }
        //-----
        private void DgvCarList_CellDoubleClick(object sender, DataGridViewCellEventArgs e){
            frmCarDetails frmCarDetails = new frmCarDetails{
                Text = "Update car",
                InsertOrUpdate = true,
                CarInfo = GetCarObject(),
                CarRepository = carRepository
            };
            if (frmCarDetails.ShowDialog() == DialogResult.OK){
                LoadCarList();
                //Set focus car updated
                source.Position = source.Count - 1;
            }
        }

        //Clear text on TextBoxes
        private void ClearText(){
            txtCarID.Text = string.Empty;
            txtCarName.Text = string.Empty;
            txtManufacturer.Text = string.Empty;
            txtPrice.Text = string.Empty;
            txtReleaseYear.Text = string.Empty;
        }
        //-----
        private Car GetCarObject()
        {
            Car car = null;
            try
            {
                car = new Car
                {
                    CarID = int.Parse(txtCarID.Text),
                    CarName = txtCarName.Text,
                    Manufacturer = txtManufacturer.Text,
                    Price = decimal.Parse(txtPrice.Text),
                    ReleaseYear = int.Parse(txtReleaseYear.Text)
                };
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Get car");
            }
            return car;
        }
    } //end GetCarObject
    //-----
}
```

```

public void LoadCarList(){
    var cars = carRepository.GetCars();
    try {
        //The BindingSource component is designed to simplify
        //the process of binding controls to an underlying data source
        source = new BindingSource();
        source.DataSource = cars;

        txtCarID.DataBindings.Clear();
        txtCarName.DataBindings.Clear();
        txtManufacturer.DataBindings.Clear();
        txtPrice.DataBindings.Clear();
        txtReleaseYear.DataBindings.Clear();

        txtCarID.DataBindings.Add("Text", source, "CarID");
        txtCarName.DataBindings.Add("Text", source, "CarName");
        txtManufacturer.DataBindings.Add("Text", source, "Manufacturer");
        txtPrice.DataBindings.Add("Text", source, "Price");
        txtReleaseYear.DataBindings.Add("Text", source, "ReleaseYear");

        dgvCarList.DataSource = null;
        dgvCarList.DataSource = source;
        if (cars.Count() == 0){
            ClearText();
            btnDelete.Enabled = false;
        }
        else{
            btnDelete.Enabled = true;
        }
    }
    catch (Exception ex) {
        MessageBox.Show(ex.Message, "Load car list");
    }
}

//end LoadCarList

//-----
private void btnLoad_Click(object sender, EventArgs e)
{
    LoadCarList();
}

//-----
private void btnClose_Click(object sender, EventArgs e) => Close();
//-----
private void btnNew_Click(object sender, EventArgs e) {
    frmCarDetails frmCarDetails = new frmCarDetails {
        Text = "Add car",
        InsertOrUpdate = false,
        CarRepository = carRepository
    };
    if(frmCarDetails.ShowDialog() == DialogResult.OK) {
        LoadCarList();
        //Set focus car inserted
        source.Position = source.Count - 1;
    }
}

//-----

```

```
private void btnDelete_Click(object sender, EventArgs e)
{
    try
    {
        var car = GetCarObject();
        carRepository.DeleteCar(car.CarID);
        LoadCarList();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Delete a car");
    }
} //end btnDelete_Click
} //end Form
}
```

Activity 04: Run the AutomobileWinApp project and test all actions

Step 01. Right-click on **AutomobileWinApp** project, select Set as Startup Project then press run the project (press Ctrl+F5 or F5)

Step 02. Click **Load** button and display the result as the below figure.

Car Management

Car ID: 1 Price: 30000

Car Name: CRV Released Year: 2021

Manufacturer: Honda

Load New Delete

	CarID	CarName	Manufacturer	Price	ReleaseYear
▶	1	CRV	Honda	30000	2021
	2	Ford Focus	Ford	15000	2020
*					

Close

Step 03. Click **New** button and display the result as the below figure, enter the values on TextBoxes then click **Save**

Car Management

Car ID: Car Name: Manufacturer:

CarID: 1, 2, ... ReleaseYear: 2021, 2020, ...

Add car

Car ID: Car Name: Manufacturer: Price: Released Year:

Save Cancel

Close

Car Management

Car ID: Price: Car Name: Released Year: Manufacturer:

Load New Delete

CarID	CarName	Manufacturer	Price	ReleaseYear
1	CRV	Honda	30000	2021
2	Ford Focus	Ford	15000	2020
3	Q8	Audi	100000000	2022
			100000000	

Close

Step 04. Select a row on the DataGridView then click **Delete** to remove a Car

Car Management

Car ID: Price:

Car Name: Released Year:

Manufacturer:

CarID	CarName	Manufacturer	Price	ReleaseYear
1	CRV	Honda	30000	2021
2	Ford Focus	Ford	15000	2020

Step 05. Double-click a row on the DataGridView to update a Car on the popup form, edit values then click **Save**

Car Management

Car ID: Price:

Car Name: Manufacturer:

CarID	CarName
1	CRV
2	Ford Focus

Update car

Car ID:

Car Name:

Manufacturer:

Price:

Released Year:

Car Management

Car ID

2

Price

150009999

Car Name

Ford Everest

Released Year

2021

Manufacturer

Ford

Load

New

Delete

	CarID	CarName	Manufacturer	Price	ReleaseYear
	1	CRV	Honda	30000	2021
▶	2	Ford Everest	Ford	150009999	2021
•					

Close