# CST-250 Activity 1: Car Store Application Guide

## Contents


Overview ..... 2
Part 1 - Car Class Library ..... 3
Part 2 - Console Application ..... 8
Console App Coding Challenge ..... 13
Part 3 – GUI Car Store ..... 14
Form Design ..... 14
Button Functions ..... 20
Challenges ..... 21

## Overview

**Objective**: Create two versions of an application that utilize the same back-end classes and data sources.

The point of this exercise is to demonstrate the separation of classes that store application data from the menus and controls of the user interface.

**Details**: We will build three separate projects:

(1) Class Library

(2) Console App

(3) Windows Forms app

(1) **Class Library** - Contains the objects and lists (inventory and shopping cart). The class library will be used in both versions of the Car Store application: Console and GUI.

a. Car Class – Contains the properties of a car (make, model, and price).

b. Store Class – Contains the methods to add new cars to the store inventory and shopping cart.

(2) **Console App** – A loop to ask the user to

a. Create new cars for the inventory

b. Place cars into the shopping cart

c. Checkout and get a final price

(3) **Windows Forms app** – A graphical user interface that accomplishes the same goals as the console app.

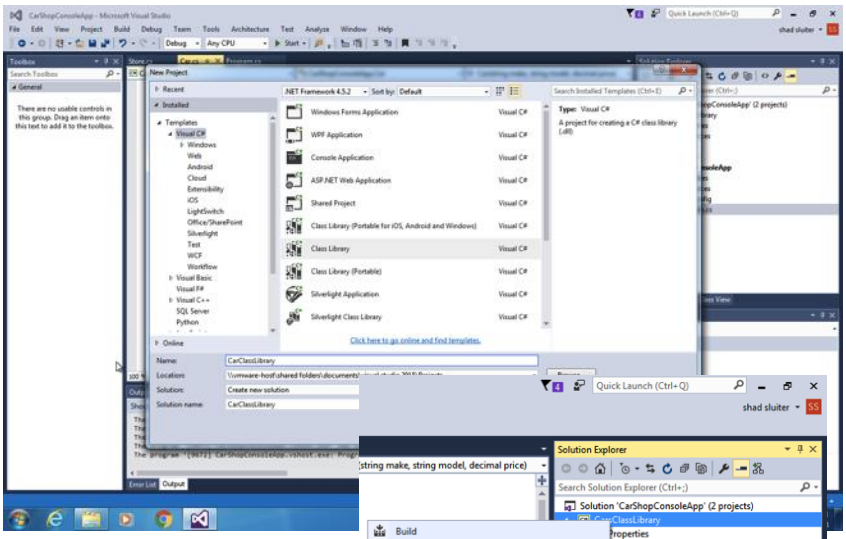
a. Create new cars for the inventory

b. Place cars into the shopping cart

c. Check out and get a final price

## Part 1 - Car Class Library

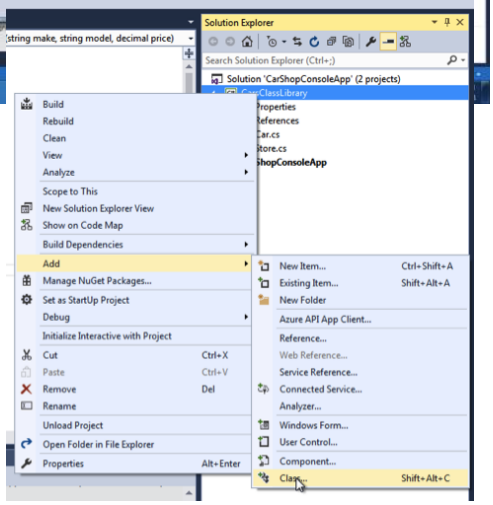
**Objective:** Create a class in C# that will be usable as a back-end for two separate versions of the app, a console version and a GUI version.



**Setup**

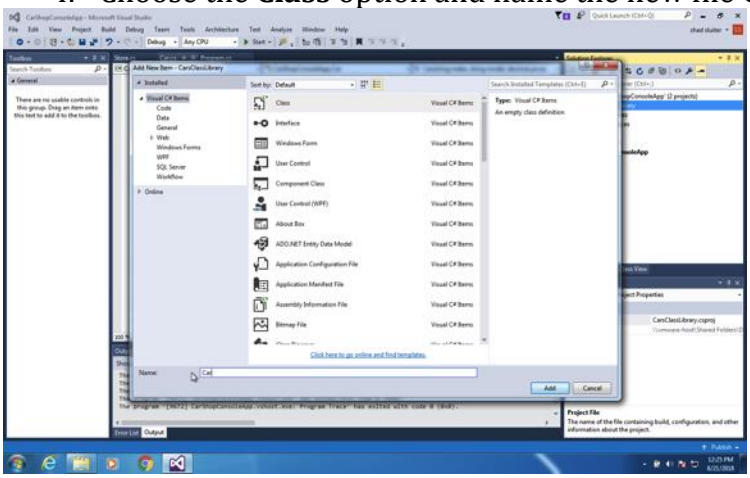
1. Start a new project in Visual Studio. Choose the **Class Library** option. Name the project **CarClassLibrary**.

2. You may delete the **class1.cs** file in the Solution Explorer.




3. Create a new class. In the Solution Explorer, right-click the **CarClassLibrary** title, choose Add → Class.

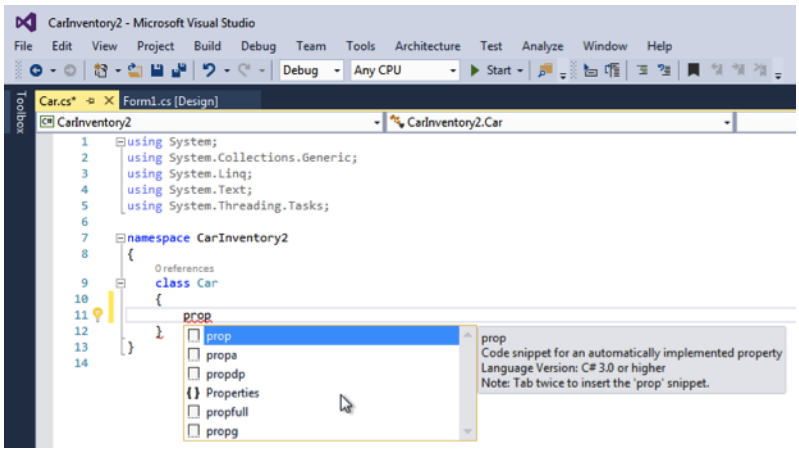
4. Choose the **Class** option and name the new file **Car**.



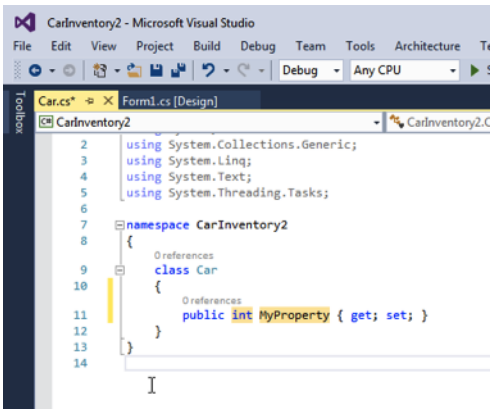
5. Create three properties to describe a car: **Make, Model, and Price**. Later, we will create additional properties such as Rating, Year, SafetyLevel, etc.

Typing shortcuts to create the class:

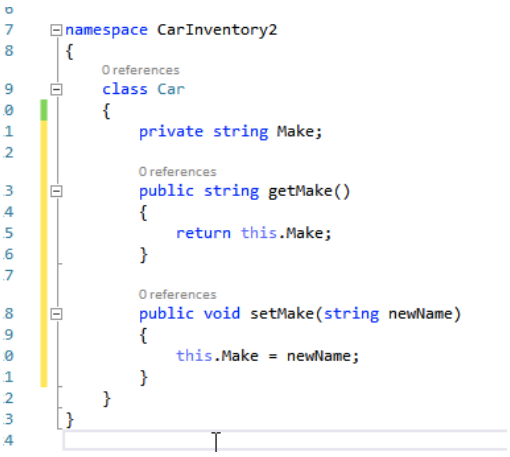
Type **prop** and wait for the snippet menu to appear…



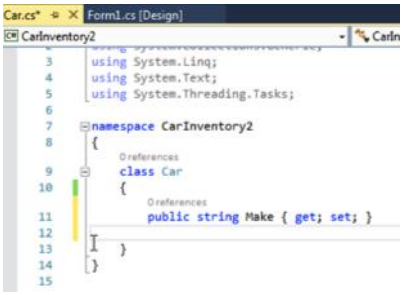

Press TAB twice and a new property is created with the shortcut “get” and “set” notation.




This property and the getter and setter can also be written like the following, which would function equally well. This might be called the “Java” way of creating properties and getters.




I am going to use the C# shortened format. Rename the data type and property name to **string Make**.




6. Create two more properties and getters and setters for the three properties. I am showing the shorthand {get; set; }.

```
namespace CarShopConsoleApp
{
    public class Car
    {
        // list the properties
        public string Make { get; set; }
        public string Model { get; set; }
        public decimal Price { get; set; }
```

7. Write the word “public” in front of the class Car line as shown. This will allow other classes to access the properties and methods inside the class.

```
namespace CarShopConsoleApp
{
    public class Car
    {
        // list the properties
        public string Make { get; set; }
        public string Model { get; set; }
        public decimal Price { get; set; }

        // Car constructor with 3 parameters
        public Car(string make, string model, decimal price)
        {
            Make = make;
            Model = model;
            Price = price;
        }

        // Car constructor with 0 parameters. Provide default values.
        public Car()
        {
            Make = "Nothing yet";
            Model = "Nothing yet";
            Price = 0;
        }
```

**Create Two Constructors:**

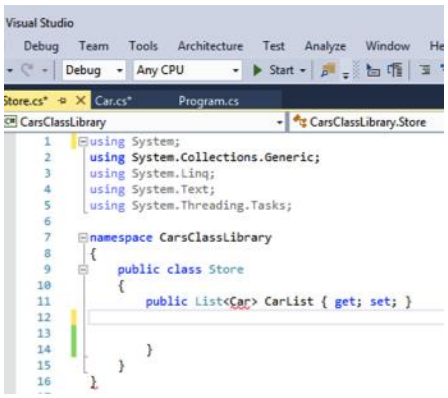
Constructor functions are called when a new instance of the class is created. We are going to create two versions of the constructor. Copy the code shown here.

The first version of the constructor accepts three parameters. The second version is a constructor with no parameters. Later in the program, we will see code like this, which will use the first version of the constructor.

```
Car myNewSportsCar = new Car(“Ford”, “Mustang”, 19500);
```

Or using the second constructor:

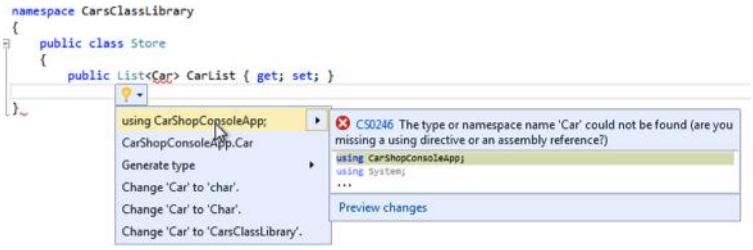
```
Car genericCar = new Car();
```



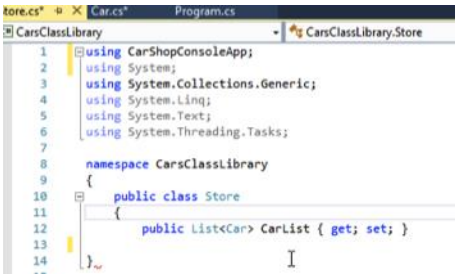

**Create a Store class**

1. Right-click in the CarsClassLibrary and choose Add → Class.
2. Name the new class Store.
3. Set the class to public.
4. Add a List<Car> CarList property to the store as shown.

Notice that the Car object is unknown. The word Car is underlined in red. Hover over the underlined word "Car" and choose "Show Potential Fixes." Visual Studio recommends that you add the words "using CarShopConsoleApp" to the first line of the class.




Notice that there is a new statement on line #1. Adding the new line removes the error. The Store class now is aware of other classes in the same project.




Create a second list for shopping the shopping cart list.

```
namespace CarsClassLibrary
{
    public class Store
    {
        public List<Car> CarList { get; set; }
        public List<Car> ShoppingList { get; set; }
```

Create a constructor for the store. This does the important work of initializing the values of the two lists as an empty list. Failure to assign a value to the lists will result in a run-

```
namespace CarsClassLibrary
{
    public class Store
    {
        public List<Car> CarList { get; set; }
        public List<Car> ShoppingList { get; set; }

        public Store()
        {
            // We must initilize each list or see the dreaded error: "Null Reference Exception Object
              reference not set to an instance of an object."
            CarList = new List<Car>();
            ShoppingList = new List<Car>();
        }
```

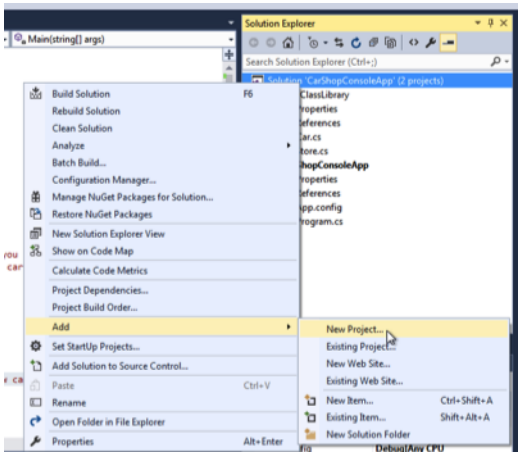
time error “Null Reference Exception”.

**Checkout Function**

Create another method that will calculate the total value of the items in the shopping cart list. We need to loop through each item in the cart and add its price to the total bill. After calculating the total cost, the shopping cart is emptied.

```
namespace CarsClassLibrary
{
    public class Store
    {
        public List<Car> CarList { get; set; }
        public List<Car> ShoppingList { get; set; }

        public Store()...

        public decimal checkout()
        {
            decimal totalCost = 0;

            // calculate the total cost of items in the cart.
            foreach (var c in ShoppingList)
            {
                totalCost += c.Price;
            }
            // clear the shopping cart
            ShoppingList.Clear();

            // return the total
            return totalCost;

        }
    }
}
```

## Part 2 - Console Application

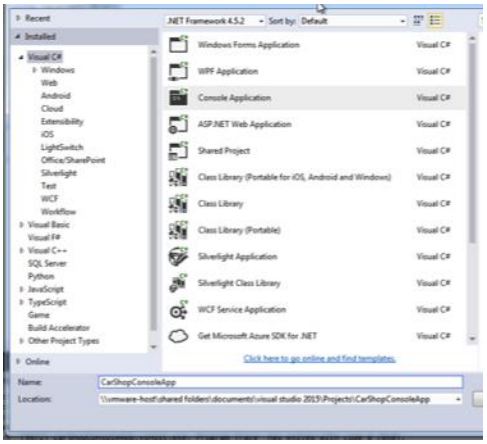



In this section, we will create an application that will utilize the two classes we just created.

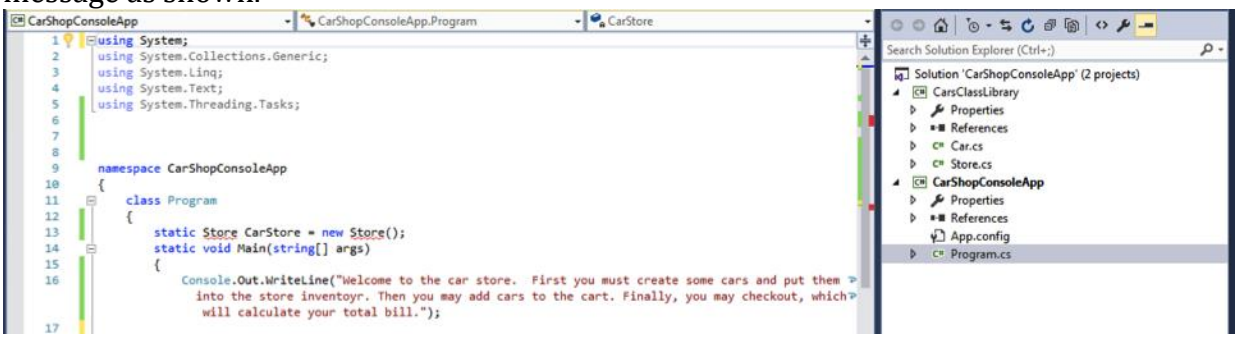
Create a new console application project.

1. Right-click on the **CarShopConsoleApp** line and choose **Add → New Project**.

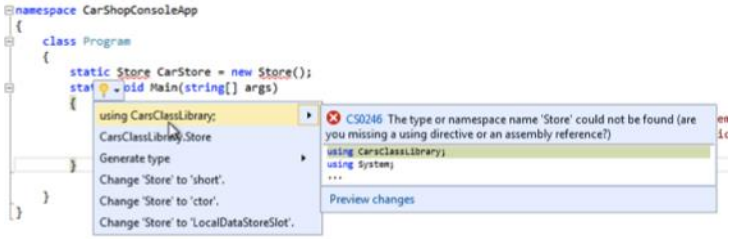
2. Choose **Console Application** and name the project **CarShopConsoleApp**




In the **Program.cs** file, try to add a new instance to the Store class and print a welcome message as shown.




**Static**
Notice that we have to use the word "Static" for our Store class because the Main(string[] args) class is static.

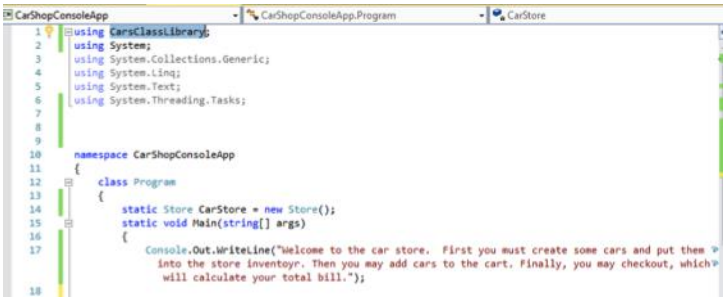



**Unknown class**
Notice that the **Store** class is unknown to this project.

Hover over the Store class and select the suggested fix.

Adding the new using statement on line 1 removes the error, using CarClassLibrary.




## The Control Loop

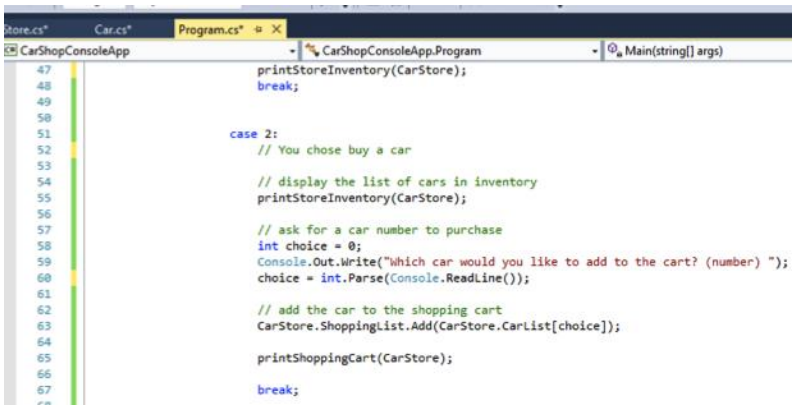
Next, we will create a mechanism to control the flow of the program. We will prompt the user to respond with 0, 1, 2, or 3 for valid actions. Continue the loop until a 0 is entered. Notice this code has no error checking for invalid input. That will have to be corrected later.

```
namespace CarShopConsoleApp
{
    class Program
    {
        static Store CarStore = new Store();
        static void Main(string[] args)
        {
            Console.Out.WriteLine("Welcome to the car store.  First you must create some cars and put them into
              the store inventoyr. Then you may add cars to the cart. Finally, you may checkout, which will
              calculate your total bill.");
            int action = chooseAction();
            while (action != 0)
            {
               // add case stements and actions here.

                action = chooseAction();
            }
        }

        static public int chooseAction()
        {
            int choice = 0;
            Console.Out.Write("Choose an action (0) quit (1) add a car (2) add item to cart (3) checkout ");
            choice = int.Parse( Console.ReadLine());
            return choice;
        }
    }
}
```

Implement the **Add a Car** option. This is the start of the switch statement. If a user enters “1” for the action, we prompt the user to create a new car object and add it to the store’s car list. Notice there is no error checking in this code either.

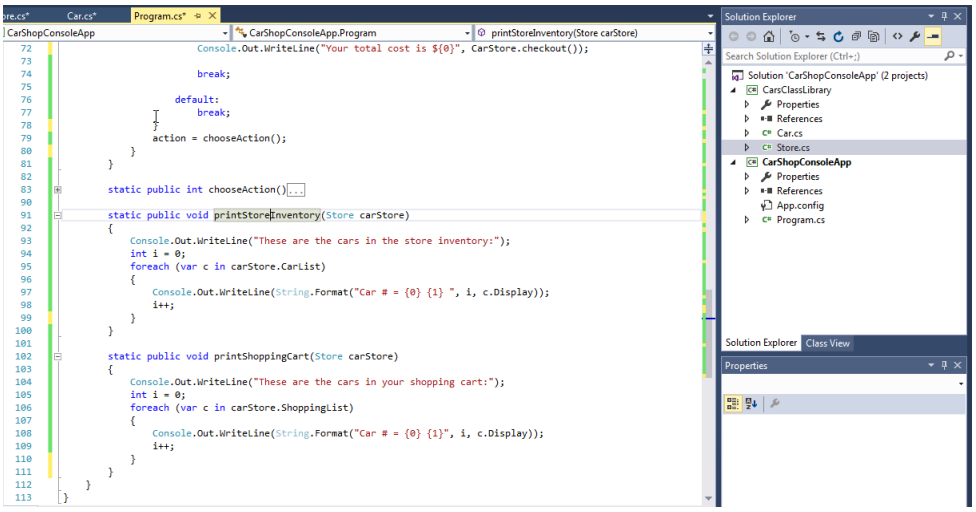
```
CarShopConsoleApp.Program
Main(string[] args)
{
    Console.Out.WriteLine("Welcome to the car store.  First you must create some cars and put them into
      the store inventoyr. Then you may add cars to the cart. Finally, you may checkout, which will
      calculate your total bill.");
    int action = chooseAction();
    while (action != 0)
    {
        switch (action)
        {
            case 1:
                // You chose add a car
                Console.Out.WriteLine("You chose to add a new car to the store:");

                // ask for three property details
                String carMake = "";
                String carModel = "";
                Decimal carPrice = 0;

                Console.Out.Write("What is the car make?  Ford, GM, Toyota etc ");
                carMake = Console.ReadLine();

                Console.Out.Write("Waht is the car model?  Corvette, Focus, Ranger ");
                carModel = Console.ReadLine();

                Console.Out.Write("What is the car price?  Only numbers please ");
                carPrice = int.Parse(Console.ReadLine());

                // create a new car object and add it to the list
                Car newCar = new Car();
                newCar.Make = carMake;
                newCar.Model = carModel;
                newCar.Price = carPrice;
                CarStore.CarList.Add(newCar);
                printStoreInventory(CarStore);
                break;
```

Implement **Add to Cart** (option #2). This function relies on a helper function called **printStoreInventory** which is shown below.



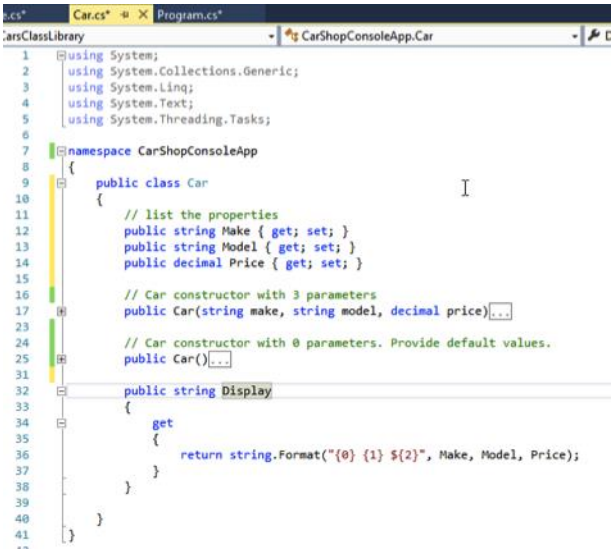
```
                    printStoreInventory(CarStore);
                    break;


                case 2:
                    // You chose buy a car

                    // display the list of cars in inventory
                    printStoreInventory(CarStore);

                    // ask for a car number to purchase
                    int choice = 0;
                    Console.Out.Write("Which car would you like to add to the cart? (number) ");
                    choice = int.Parse(Console.ReadLine());

                    // add the car to the shopping cart
                    CarStore.ShoppingList.Add(CarStore.CarList[choice]);

                    printShoppingCart(CarStore);

                    break;
```

Here is the code for the helper function, **printStoreInventory** and **printShoppingCart.** Place this function somewhere below. I placed it after the **chooseAction**() function as shown here.



```
                    Console.Out.WriteLine("Your total cost is ${0}", CarStore.checkout());

                    break;

                default:
                    break;
                }
                action = chooseAction();
            }
        }

        static public int chooseAction()...

        static public void printStoreInventory(Store carStore)
        {
            Console.Out.WriteLine("These are the cars in the store inventory:");
            int i = 0;
            foreach (var c in carStore.CarList)
            {
                Console.Out.WriteLine(String.Format("Car # = {0} {1} ", i, c.Display));
                i++;
            }
        }

        static public void printShoppingCart(Store carStore)
        {
            Console.Out.WriteLine("These are the cars in your shopping cart:");
            int i = 0;
            foreach (var c in carStore.ShoppingList)
            {
                Console.Out.WriteLine(String.Format("Car # = {0} {1}", i, c.Display));
                i++;
            }
        }
    }
}
```

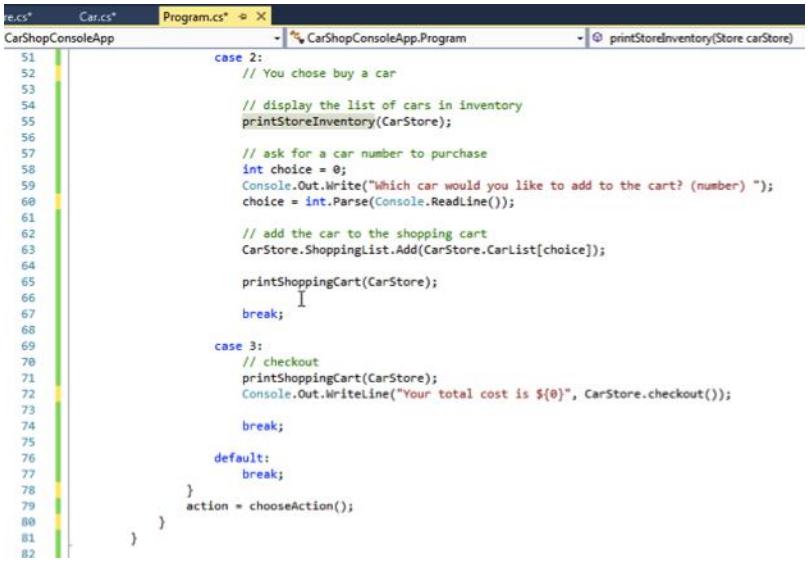
**What is c.Display???**
The printStoreInventory function uses a property in **Car.cs** that we haven't created yet! **Display** is a property derived from the other properties in Car that is used to print all three properties under one name. Go back to the **Car** class and add the following code on lines #32–40. You could also simply override the **toString()** method and get the same results as Display.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CarShopConsoleApp
{
    public class Car
    {
        // list the properties
        public string Make { get; set; }
        public string Model { get; set; }
        public decimal Price { get; set; }

        // Car constructor with 3 parameters
        public Car(string make, string model, decimal price)...

        // Car constructor with 0 parameters. Provide default values.
        public Car()...

        public string Display
        {
            get
            {
                return string.Format("{0} {1} ${2}", Make, Model, Price);
            }
        }

    }
}
```

**Checkout**

Finally, we create the code for the third action in our application, **checkout**.




**Default**

Every switch and case block needs to have a default action. The default action will handle the case when the user types in an invalid action number such as 99 or -3. The program will crash if the user types in a letter instead of a number for their desired action. Fix this later.

**All of the code**

In case you misplaced a few items, here is the entire list of code for the application.

```
using CarsClassLibrary;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;



namespace CarShopConsoleApp
{
    class Program
    {
        static Store CarStore = new Store();
        static void Main(string[] args)
        {
            Console.Out.WriteLine("Welcome to the car store.  First you must create some cars and put them into
              the store inventoyr. Then you may add cars to the cart. Finally, you may checkout, which will
              calculate your total bill.");
            int action = chooseAction();
            while (action != 0)
            {
                switch (action)
                {
                    case 1:
                        // You chose add a car
                        Console.Out.WriteLine("You chose to add a new car to the store:");

                        // ask for three property details
                        String carMake = "";
                        String carModel = "";
                        Decimal carPrice = 0;

                        Console.Out.Write("What is the car make?  Ford, GM, Toyota etc ");
                        carMake = Console.ReadLine();


                        Console.Out.Write("Waht is the car model?  Corvette, Focus, Ranger ");
                        carModel = Console.ReadLine();

                        Console.Out.Write("What is the car price?  Only numbers please ");
                        carPrice = int.Parse(Console.ReadLine());

                        // create a new car object and add it to the list
                        Car newCar = new Car();
                        newCar.Make = carMake;
                        newCar.Model = carModel;
                        newCar.Price = carPrice;
                        CarStore.CarList.Add(newCar);
                        printStoreInventory(CarStore);
                        break;


                    case 2:
                        // You chose buy a car

                        // display the list of cars in inventory
                        printStoreInventory(CarStore);

                        // ask for a car number to purchase
                        int choice = 0;
                        Console.Out.Write("Which car would you like to add to the cart? (number) ");
                        choice = int.Parse(Console.ReadLine());

                        // add the car to the shopping cart
                        CarStore.ShoppingList.Add(CarStore.CarList[choice]);

                        printShoppingCart(CarStore);

                        break;
```

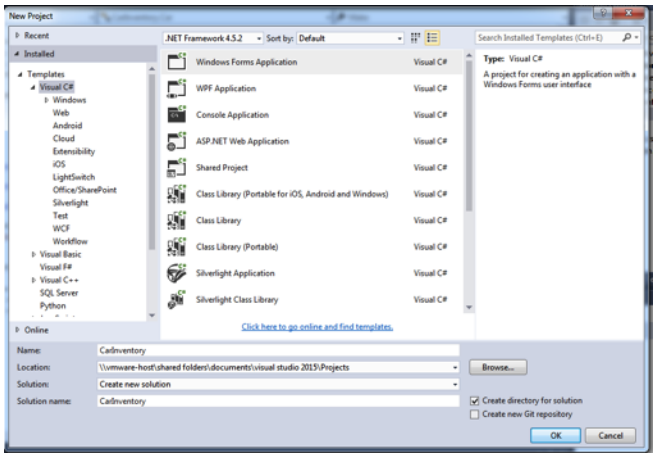
```
                case 3:
                    // checkout
                    printShoppingCart(CarStore);
                    Console.Out.WriteLine("Your total cost is ${0}", CarStore.checkout());

                    break;

                default:
                    break;
            }
            action = chooseAction();
        }
    }

    static public int chooseAction()...

    static public void printStoreInventory(Store carStore)
    {
        Console.Out.WriteLine("These are the cars in the store inventory:");
        int i = 0;
        foreach (var c in carStore.CarList)
        {
            Console.Out.WriteLine(String.Format("Car # = {0} {1} ", i, c.Display));
            i++;
        }
    }

    static public void printShoppingCart(Store carStore)
    {
        Console.Out.WriteLine("These are the cars in your shopping cart:");
        int i = 0;
        foreach (var c in carStore.ShoppingList)
        {
            Console.Out.WriteLine(String.Format("Car # = {0} {1}", i, c.Display));
            i++;
        }
    }
}
}
```

## Console App Coding Challenge

Add the following features to the program:

1) **Two new properties** – Invent two new properties for your car. You can use things like color (string), year (integer), miles (integer), isNew (boolean), sizeOfEngine (decimal), bodyCondition (integer), etc. Make the necessary property changes in the Car.cs file, as well as the rest of the program, to accommodate the new properties.
2) **Error Checking** – The console application has potential crashing points where a user can input strings in places where numbers are expected. The int.Parse( string that should be a number) function will cause exception faults unless you check the values before attempting the type conversion. Research "Try Catch C#" in Google to get some help. The user can also select numbers that are out of range of a list. Try putting car number 99 into your shopping cart and see what happens. Fix these errors so the program displays a message instead of simply crashing.

## Part 3 – GUI Car Store




In this section, we are going to build a graphic user interface (Windows app) that will have the same functions as the (ugly) console app.

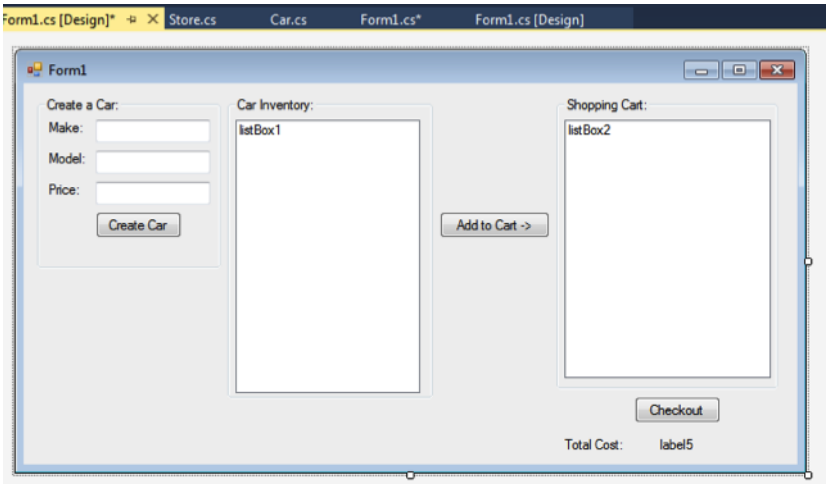
The main point of this exercise is to show the code separation between data storage and functions (backend) and the interface (console or gui). Many C# tutorials incorrectly teach that the primary place to store application data is in the form controls. Although this approach works, it is a bad pattern to follow. Good software design usually separates the class data and methods from the controls.

**Building the GUI**

1. Create a new solution. **File → New Project**.

Start a new Windows Forms Application project in Visual Studio. I am naming my project **CarShopGUI**.

## Form Design




1. For Form1, create the following controls:

GroupBox1 (Create a car)
Label1 (Make), TextBox1
Label2 (Model), TextBox2
Label3 (Price), TextBox3

GroupBox2 (Car Inventory)
ListBox1
GroupBox3 (Shopping Cart)
ListBox2

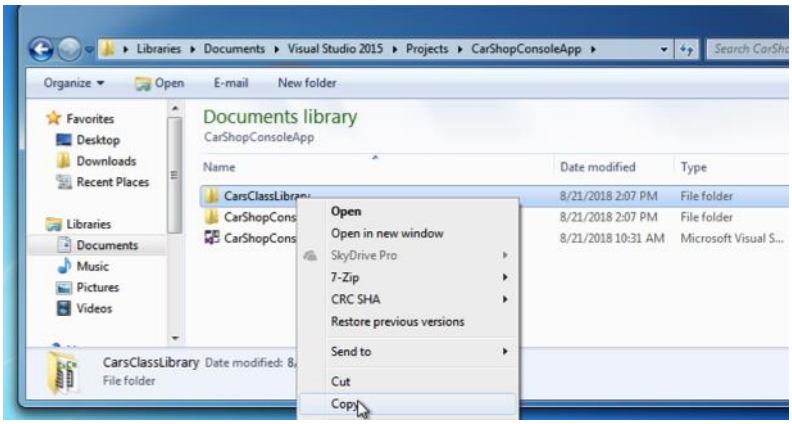
Button1 (Create Car)
Button2 (Add to Cart)
Button3 (Checkout)

Label4 (Total Cost)
Label5

Place them (approximately) in the same places as shown and change their text properties to look like the diagram.

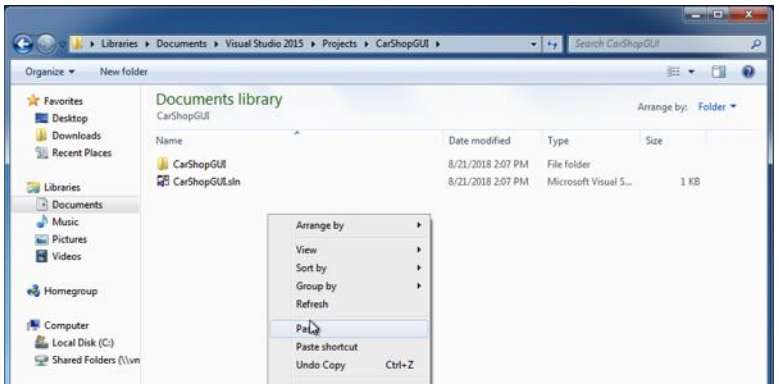
**Add the Car and Store class library to the Solution**

We will now make a copy of the project folder from our previous work.



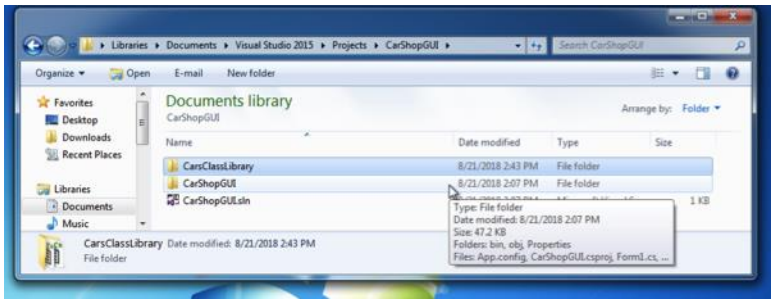

1. Open the File explorer in Windows.
2. Navigate to the folder you saved the previous work. My project was saved in

   c:\users\shadsluiter\Documents\Visual Studio 2015\Projecs\CarShopConsoleApp



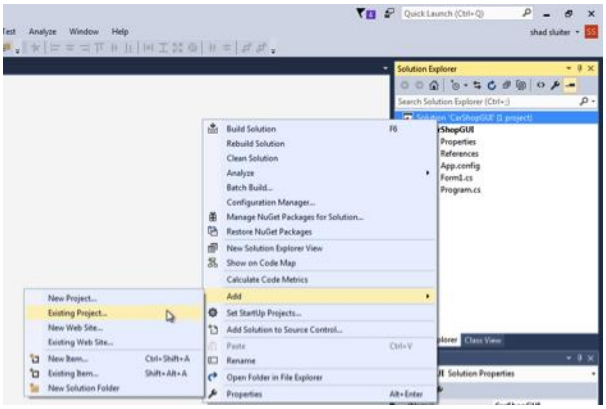

3. Right click on the **CarsClassLibrary** folder and choose **copy**.
4. Navigate up one folder and go into the **CarShopGUI** folder.
5. **Right-click** and **paste** the CarsClassLibrary here.

There should be a new folder in the window.



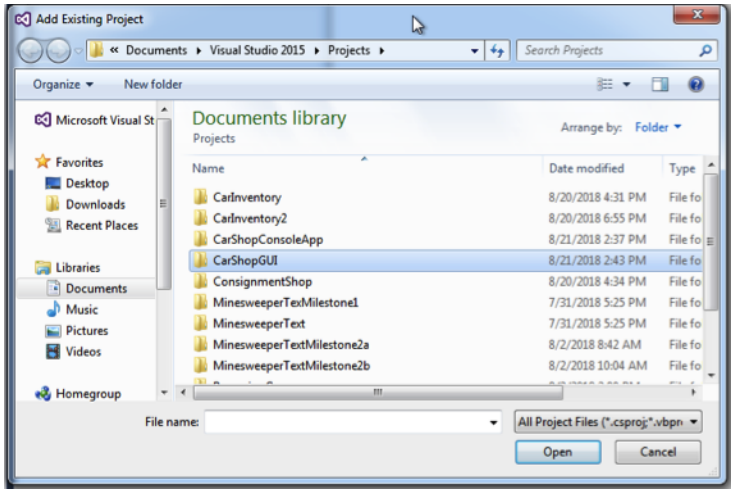

 

6. Return to Visual Studio.
7. Right-click on the solution title CarShopGUI and choose Add → Existing Project.

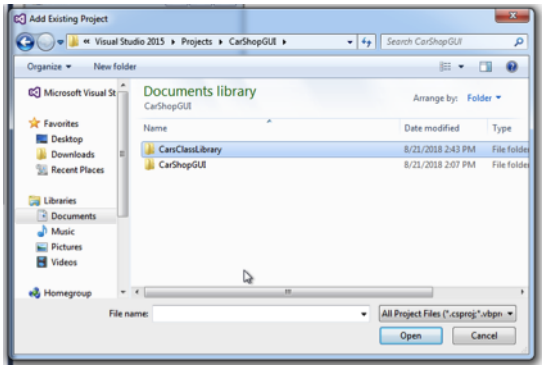


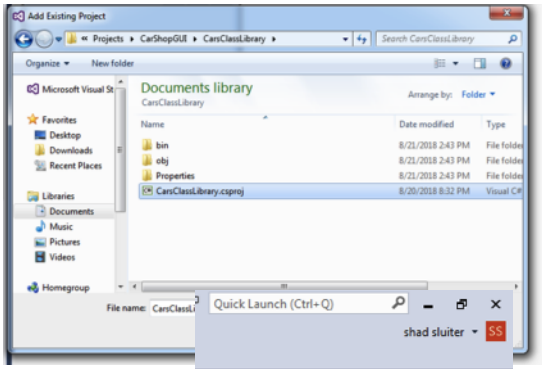

8. Open the **CarShopGUI** folder.

9. Open the **CarsClassLibrary** folder.

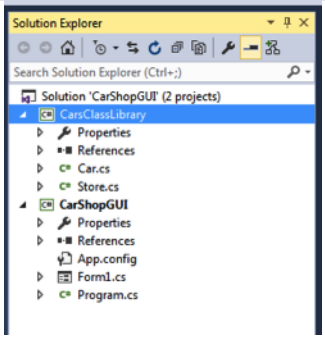



10. Open the **CarsClassLibrary.csproj** file.



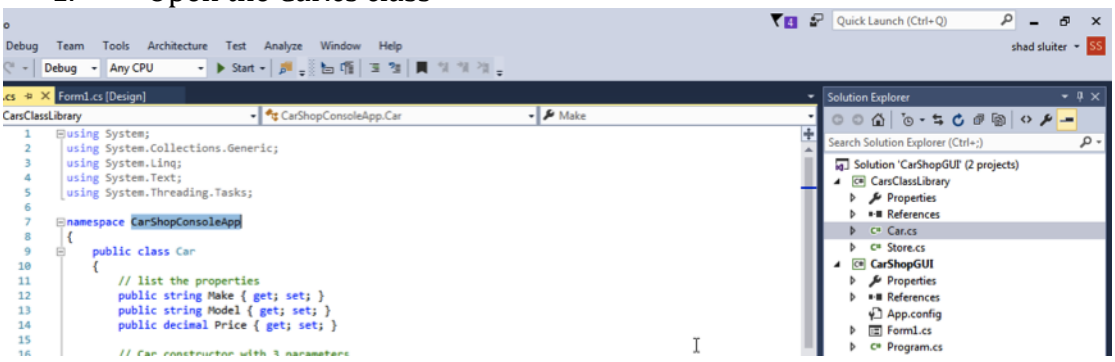



You should now see the CarsClassLibrary project has been added to your solution.

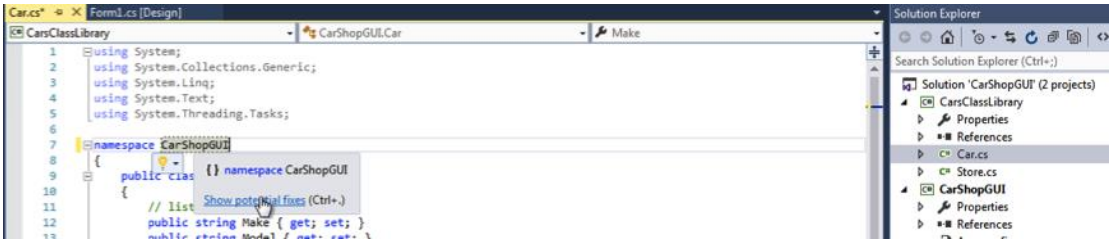



**Refactor namespaces for the Car and Store classes**

1. Open the Car.cs class

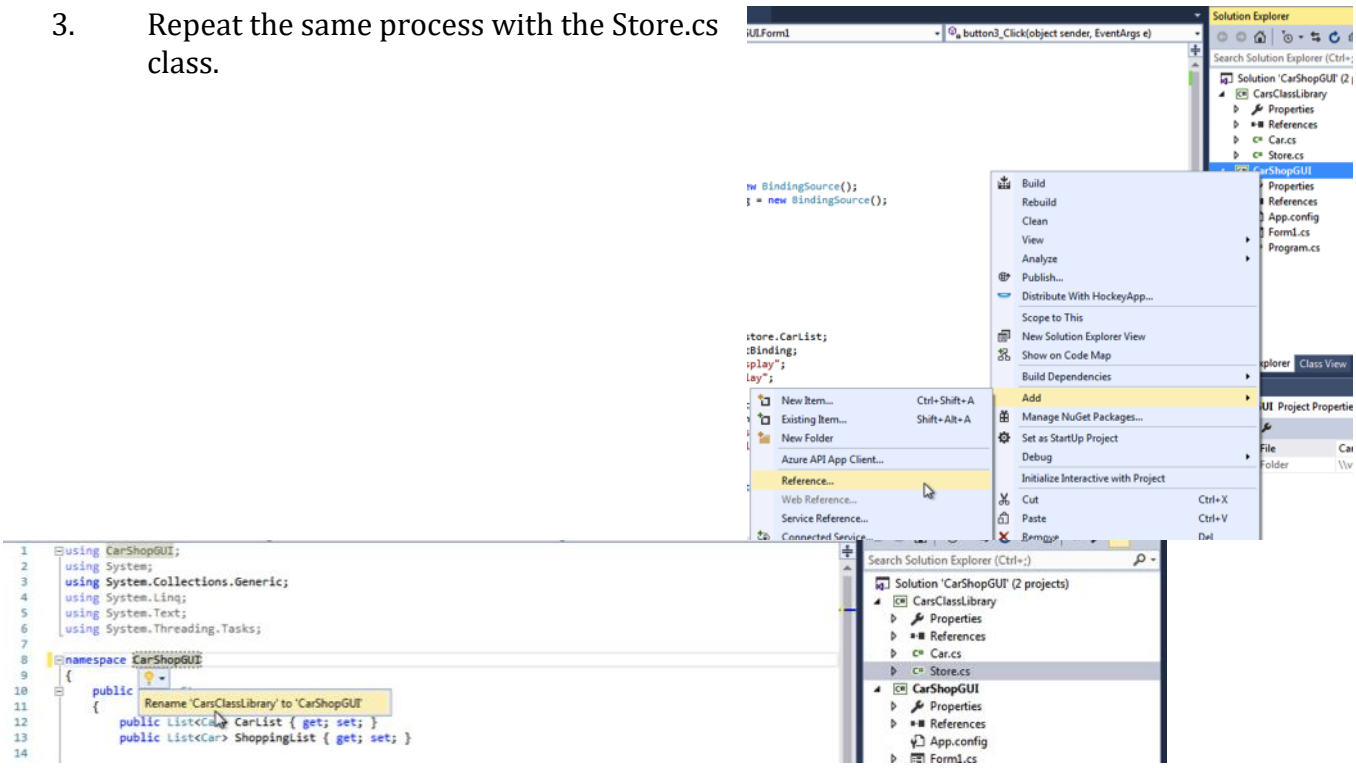



2. Change the namespace from **CarShopConsoleApp** to the new Solution name **CarShopGUI**. Visual Studio will not accept the new name until you hover over the new text and choose **Show Potential Fixes**.



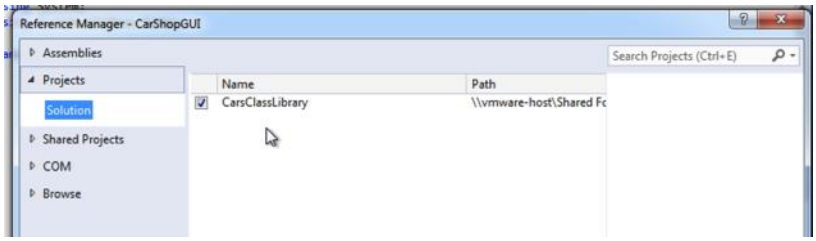

 

3. Repeat the same process with the Store.cs class.




## Connect to the two projects

1. Right-click on the CarShopGUI project name(not solution) and choose **Add → Reference**.
2. Select Projects and place a checkmark by **CarsClassLibrary**.

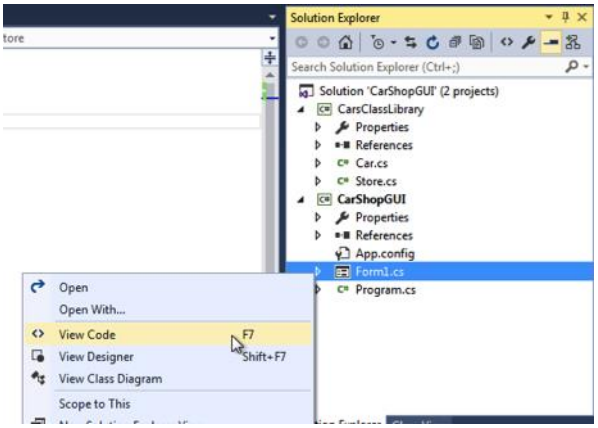



This will allow the GUI application to access the classes contained inside the CarsClassLibrary.

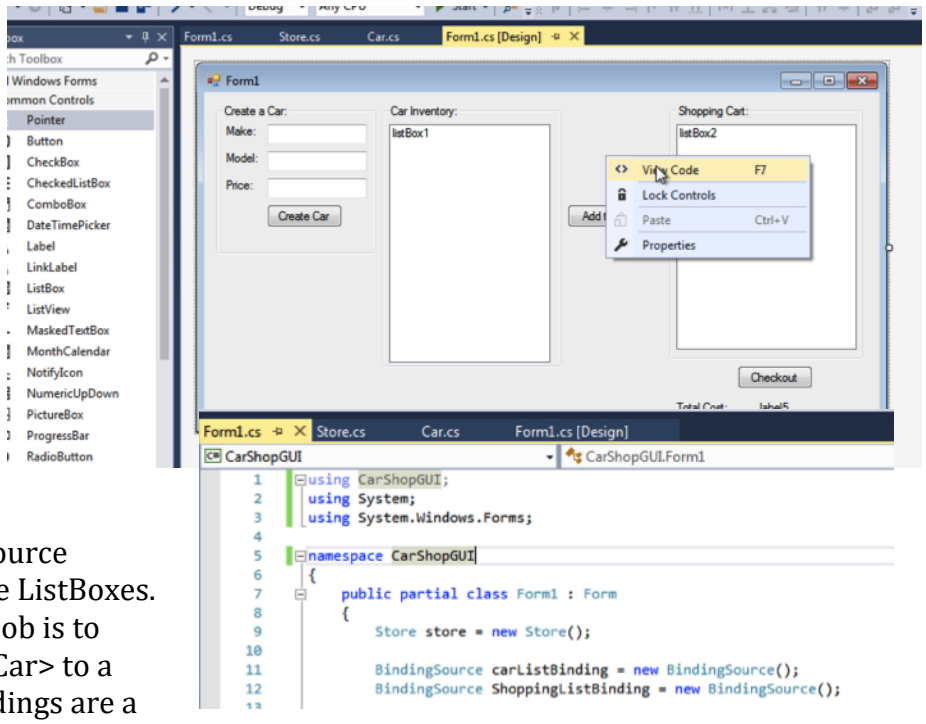
 

## Set Bindings

The next step is to connect the two lists in the Store.cs class to the form controls ListBox1 (store inventory) and ListBox2 (shopping cart).

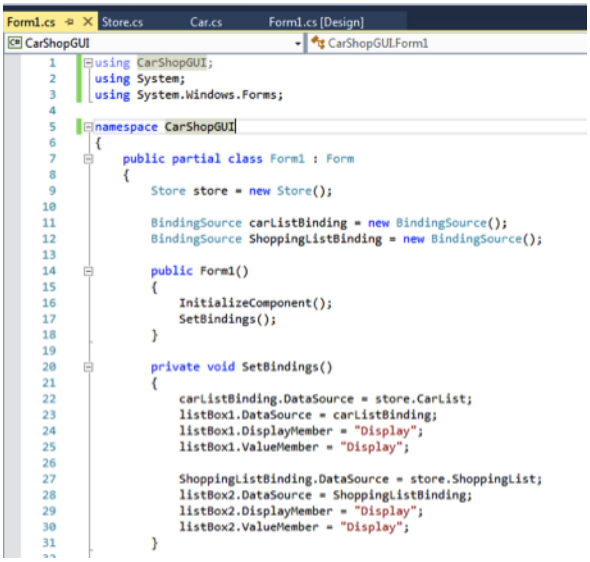



1. View the code for the form. There are several options to get to the code view. (1) Right-click on the Form1.cs file inside the Solution Explorer and choose **View Code**. (2) You can also select the Form1.cs item and press **F7**. (3) You can also right-click on the form in the design view and choose **View Code**.




2. Create two new BindingSource objects, one for each of the ListBoxes. A BindingSource object's job is to connect objects like List<Car> to a ListBox form control. Bindings are a little strange, so this would be a good time to pause and do some more research on how they are supposed to work.

3. Create a function called **SetBindings**() and fill it with the following code. This is the process for connecting the two lists in the Store class to the two listbox controls on the form.
4. Call the **SetBindings**() function from the **Form1**() constructor function as pictured here.

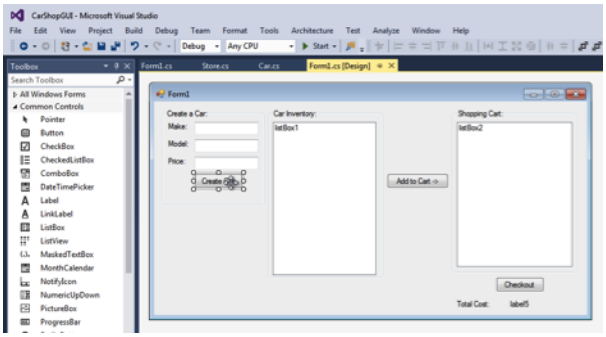


```
using CarShopGUI;
using System;
using System.Windows.Forms;

namespace CarShopGUI
{
    public partial class Form1 : Form
    {
        Store store = new Store();

        BindingSource carListBinding = new BindingSource();
        BindingSource ShoppingListBinding = new BindingSource();

        public Form1()
        {
            InitializeComponent();
            SetBindings();
        }

        private void SetBindings()
        {
            carListBinding.DataSource = store.CarList;
            listBox1.DataSource = carListBinding;
            listBox1.DisplayMember = "Display";
            listBox1.ValueMember = "Display";

            ShoppingListBinding.DataSource = store.ShoppingList;
            listBox2.DataSource = ShoppingListBinding;
            listBox2.DisplayMember = "Display";
            listBox2.ValueMember = "Display";
        }
```

listBox1.DataSource = carListBinding tells the form control where to get its data.

lsitBox1.DisplayMember = “Display” tells the control what property of the Car class to print on each line. Recall that the Display property is a combination of all three properties of the Car.

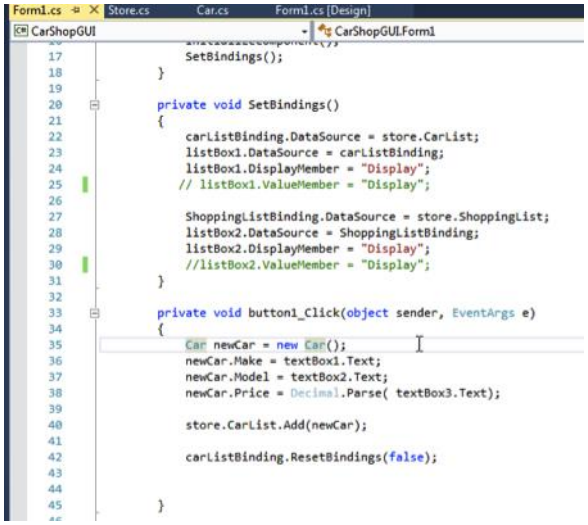
listBox1.ValueMember = “Display” is an optional line in this application.



## Button Functions

Now it is time to create the actions that make the application run.

**Button1 – Create Car**

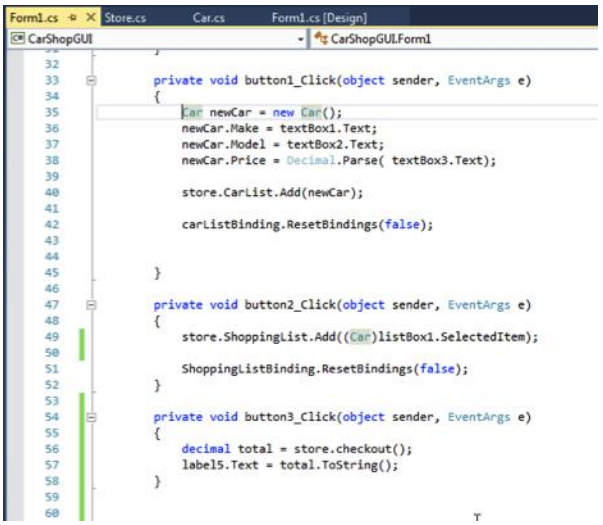


```
            SetBindings();
        }

        private void SetBindings()
        {
            carListBinding.DataSource = store.CarList;
            listBox1.DataSource = carListBinding;
            listBox1.DisplayMember = "Display";
            // listBox1.ValueMember = "Display";

            ShoppingListBinding.DataSource = store.ShoppingList;
            listBox2.DataSource = ShoppingListBinding;
            listBox2.DisplayMember = "Display";
            //listBox2.ValueMember = "Display";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Car newCar = new Car();
            newCar.Make = textBox1.Text;
            newCar.Model = textBox2.Text;
            newCar.Price = Decimal.Parse( textBox3.Text);

            store.CarList.Add(newCar);

            carListBinding.ResetBindings(false);

        }
```

1. Double click on Button1 (Create Car). This will create a click function in the Form1 code for Button1.
2. This function will do four things: (1) create a new Car object, (2) set its three properties according to the values in the three text input controls, (3) then add it to the Store CarList variable, and (4) refresh the ListBox1 control. Notice that the Price property is a decimal value. The value of a textbox is a string, so we need to change the string to a decimal. No error checking is done here. We will leave that for a future task.

Why false?

The “false” parameter in ResetBindings(false) means that the data type of the control has not changed. It still contains a list of type Car.




**Button 2 – Add to Cart**

1. Return to the Design View for Form1.
2. Double click on Button2 (Add to Cart)
3. Add the following code.

This function simply makes a copy of the currently selected Car from ListBox1 and puts it into ListBox2. The ResetBindings(false) command must be called whenever you change the contents of a ListBox.

**Button 3 – Checkout**

1. Return to the Design View for Form1.
2. Double click on Button3 (Checkout)
3. Add the code shown. The checkout function is part of the Store class.

## Challenges

Add the following features to the program:

1) **Two new properties** – Invent two new properties for your car. Make the necessary property changes in the Car.cs file, as well as the rest of the program, to accommodate the new properties. You will need to create some new controls on the form to handle the new properties.
2) **Error Checking** – The application has potential crashing points where a user can input strings in places where numbers are expected. The int.Parse( string that should be a number) function will cause exception faults unless you check the values before attempting the type conversion. Use your knowledge of C# or research “Try Catch C#” via Google to get some help.

**Deliverables**:

1) Zip file containing all source code.
2) Microsoft Word document containing screenshots of the application being run. Be sure to demonstrate all features that were created in the tutorial, as well as the challenges.