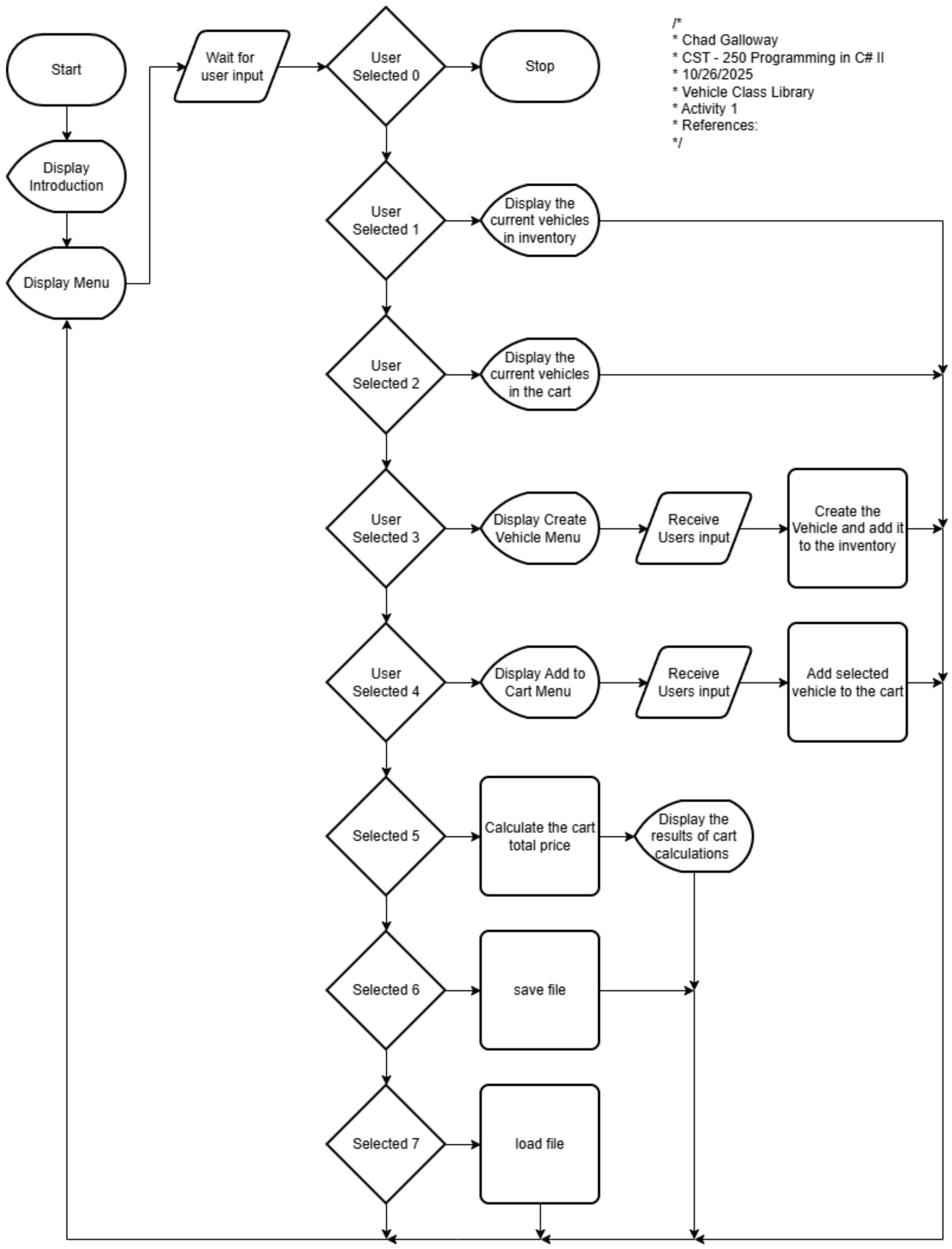
Chad Galloway
CST-250 Programming in C# II
Grand Canyon University
Oct. 26, 2025
Activity 1

Files
https://github.com/CGalloway3/CST-250-Projects/tree/master/Activity%201

Video
https://www.loom.com/share/0b708cd1a5ec482eac7e5269918f2441

Part 1

FLOW CHART


Start
Display Introduction
Display Menu
Wait for user input
User Selected 0
Stop
User Selected 1
Display the current vehicles in inventory
User Selected 2
Display the current vehicles in the cart
User Selected 3
Display Create Vehicle Menu
Receive Users input
Create the Vehicle and add it to the inventory
User Selected 4
Display Add to Cart Menu
Receive Users input
Add selected vehicle to the cart
Selected 5
Calculate the cart total price
Display the results of cart calculations
Selected 6
save file
Selected 7
load file

```
/*
* Chad Galloway
* CST - 250 Programming in C# II
* 10/26/2025
* Vehicle Class Library
* Activity 1
* References:
*/
```

Figure X: Flow chart of Activity 1

UML Class Diagram

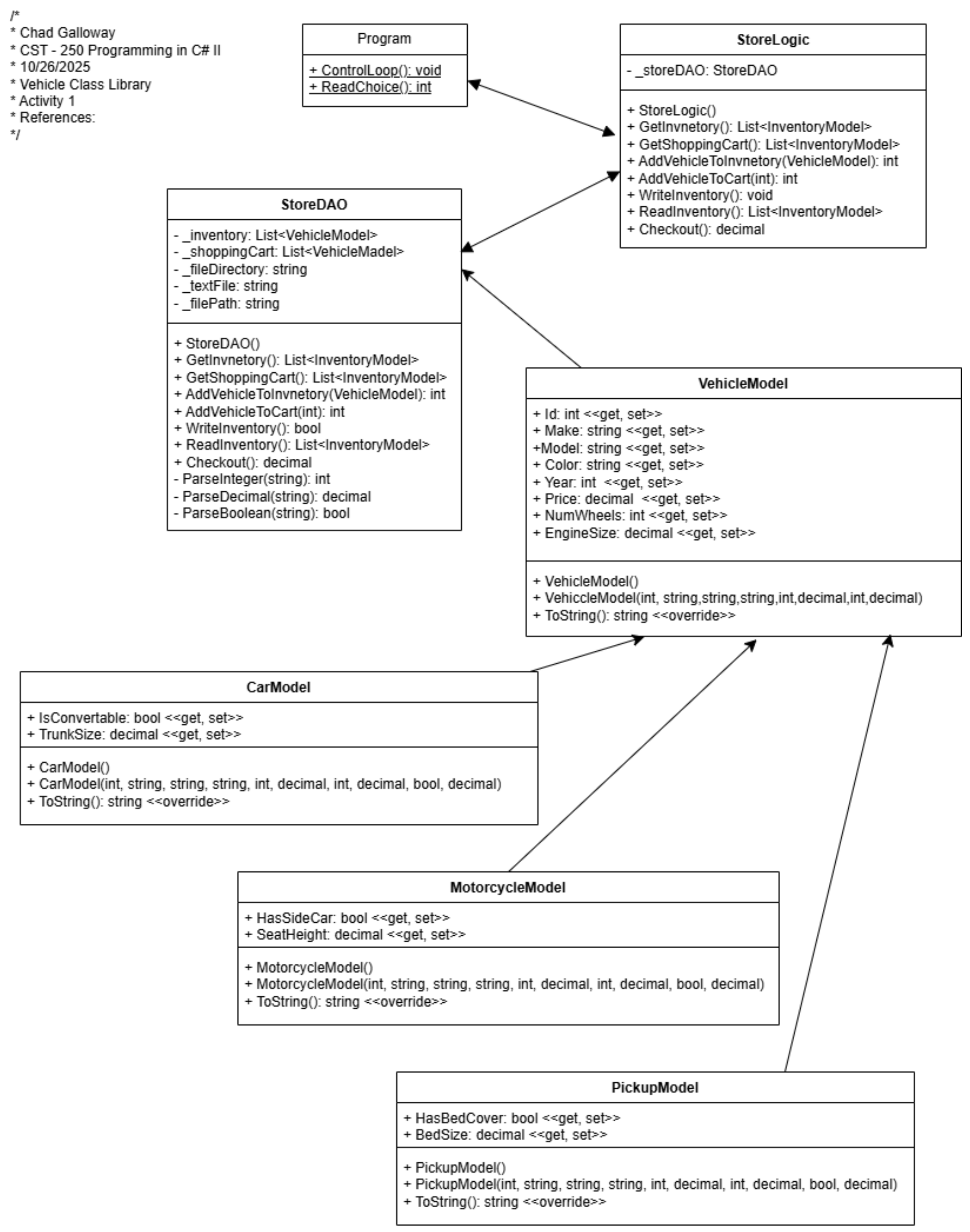
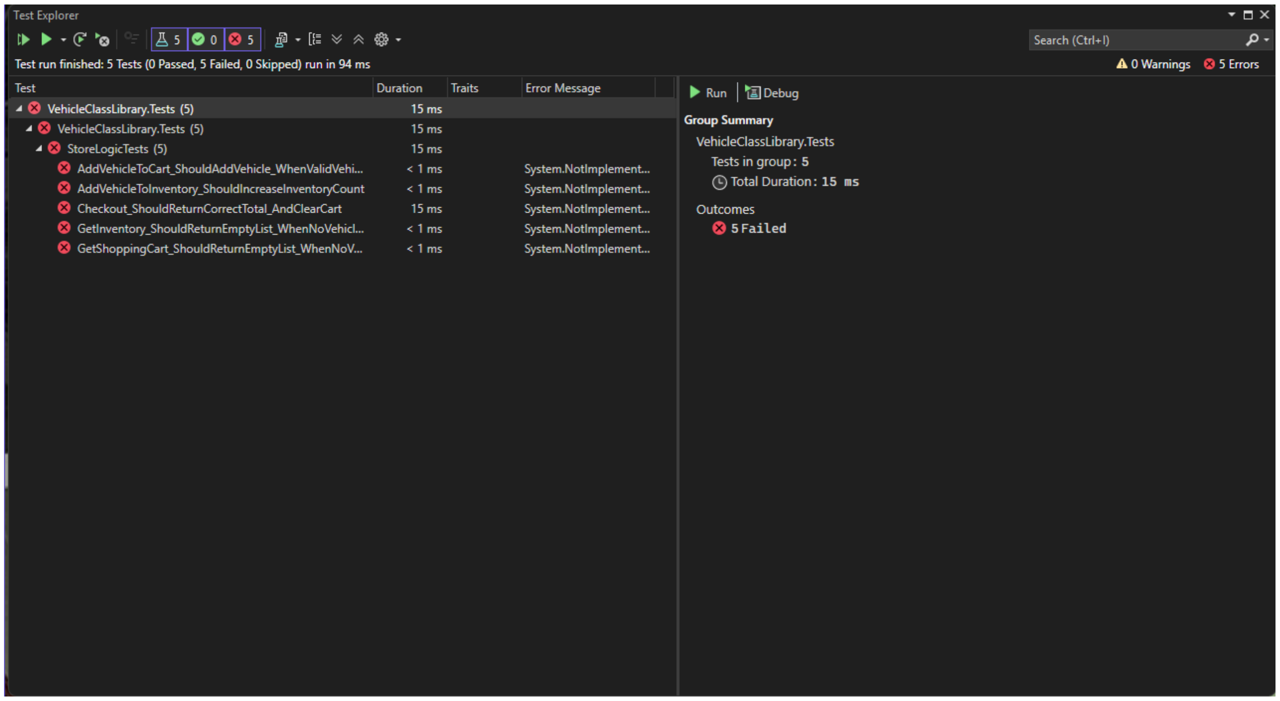

/*
* Chad Galloway
* CST - 250 Programming in C# II
* 10/26/2025
* Vehicle Class Library
* Activity 1
* References:
*/
Program
+ ControlLoop(): void
+ ReadChoice(): int
StoreLogic
- _storeDAO: StoreDAO
+ StoreLogic()
+ GetInvnetory(): List<InventoryModel>
+ GetShoppingCart(): List<InventoryModel>
+ AddVehicleToInvnetory(VehicleModel): int
+ AddVehicleToCart(int): int
+ WriteInventory(): void
+ ReadInventory(): List<InventoryModel>
+ Checkout(): decimal
StoreDAO
- _inventory: List<VehicleModel>
- _shoppingCart: List<VehicleMadel>
- _fileDirectory: string
- _textFile: string
- _filePath: string
+ StoreDAO()
+ GetInvnetory(): List<InventoryModel>
+ GetShoppingCart(): List<InventoryModel>
+ AddVehicleToInvnetory(VehicleModel): int
+ AddVehicleToCart(int): int
+ WriteInventory(): bool
+ ReadInventory(): List<InventoryModel>
+ Checkout(): decimal
- ParseInteger(string): int
- ParseDecimal(string): decimal
- ParseBoolean(string): bool
VehicleModel
+ Id: int <<get, set>>
+ Make: string <<get, set>>
+Model: string <<get, set>>
+ Color: string <<get, set>>
+ Year: int <<get, set>>
+ Price: decimal <<get, set>>
+ NumWheels: int <<get, set>>
+ EngineSize: decimal <<get, set>>
+ VehicleModel()
+ VehiccleModel(int, string,string,string,int,decimal,int,decimal)
+ ToString(): string <<override>>
CarModel
+ IsConvertable: bool <<get, set>>
+ TrunkSize: decimal <<get, set>>
+ CarModel()
+ CarModel(int, string, string, string, int, decimal, int, decimal, bool, decimal)
+ ToString(): string <<override>>
MotorcycleModel
+ HasSideCar: bool <<get, set>>
+ SeatHeight: decimal <<get, set>>
+ MotorcycleModel()
+ MotorcycleModel(int, string, string, string, int, decimal, int, decimal, bool, decimal)
+ ToString(): string <<override>>
PickupModel
+ HasBedCover: bool <<get, set>>
+ BedSize: decimal <<get, set>>
+ PickupModel()
+ PickupModel(int, string, string, string, int, decimal, int, decimal, bool, decimal)
+ ToString(): string <<override>>

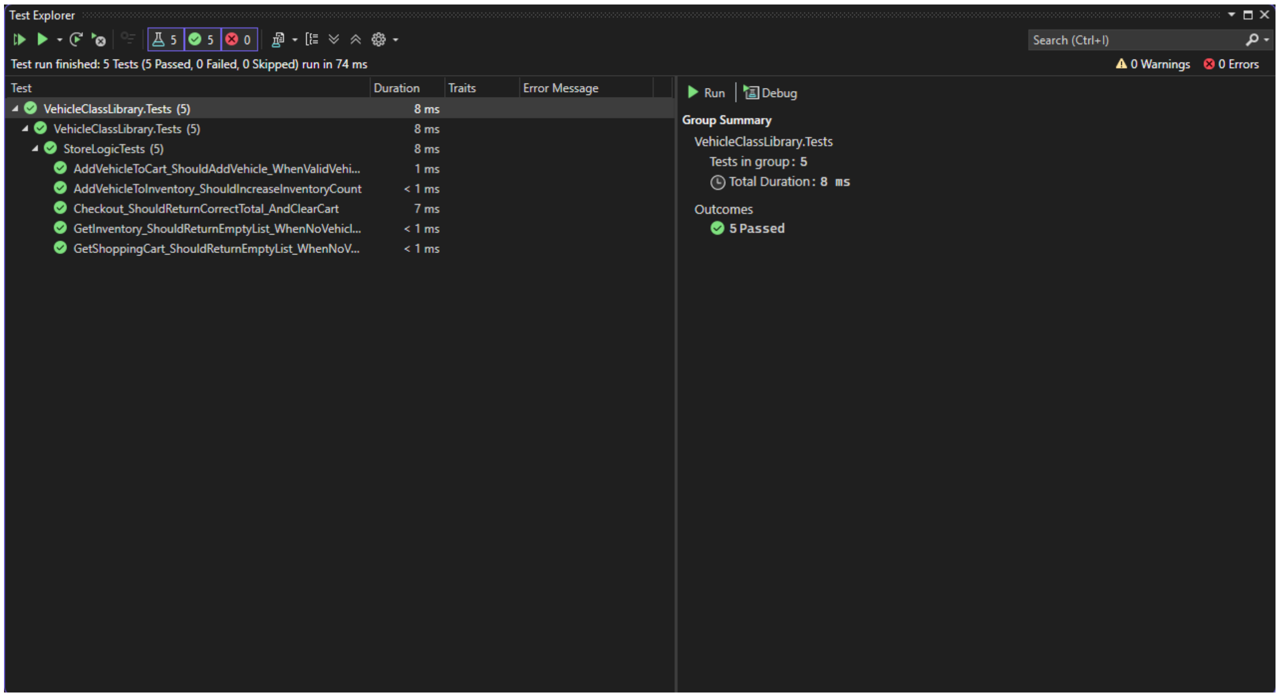
Figure X: UML Class Diagram

Screen Shots



```
Test Explorer
5  0  5                                                        Search (Ctrl+I)
Test run finished: 5 Tests (0 Passed, 5 Failed, 0 Skipped) run in 94 ms      0 Warnings   5 Errors
Test                                              Duration  Traits  Error Message
VehicleClassLibrary.Tests (5)                     15 ms
  VehicleClassLibrary.Tests (5)                   15 ms
    StoreLogicTests (5)                           15 ms
      AddVehicleToCart_ShouldAddVehicle_WhenValidVehi...   < 1 ms   System.NotImplement...
      AddVehicleToInventory_ShouldIncreaseInventoryCount   < 1 ms   System.NotImplement...
      Checkout_ShouldReturnCorrectTotal_AndClearCart       15 ms    System.NotImplement...
      GetInventory_ShouldReturnEmptyList_WhenNoVehicl...   < 1 ms   System.NotImplement...
      GetShoppingCart_ShouldReturnEmptyList_WhenNoV...     < 1 ms   System.NotImplement...

Run | Debug
Group Summary
  VehicleClassLibrary.Tests
    Tests in group: 5
    Total Duration: 15 ms
  Outcomes
    5 Failed
```

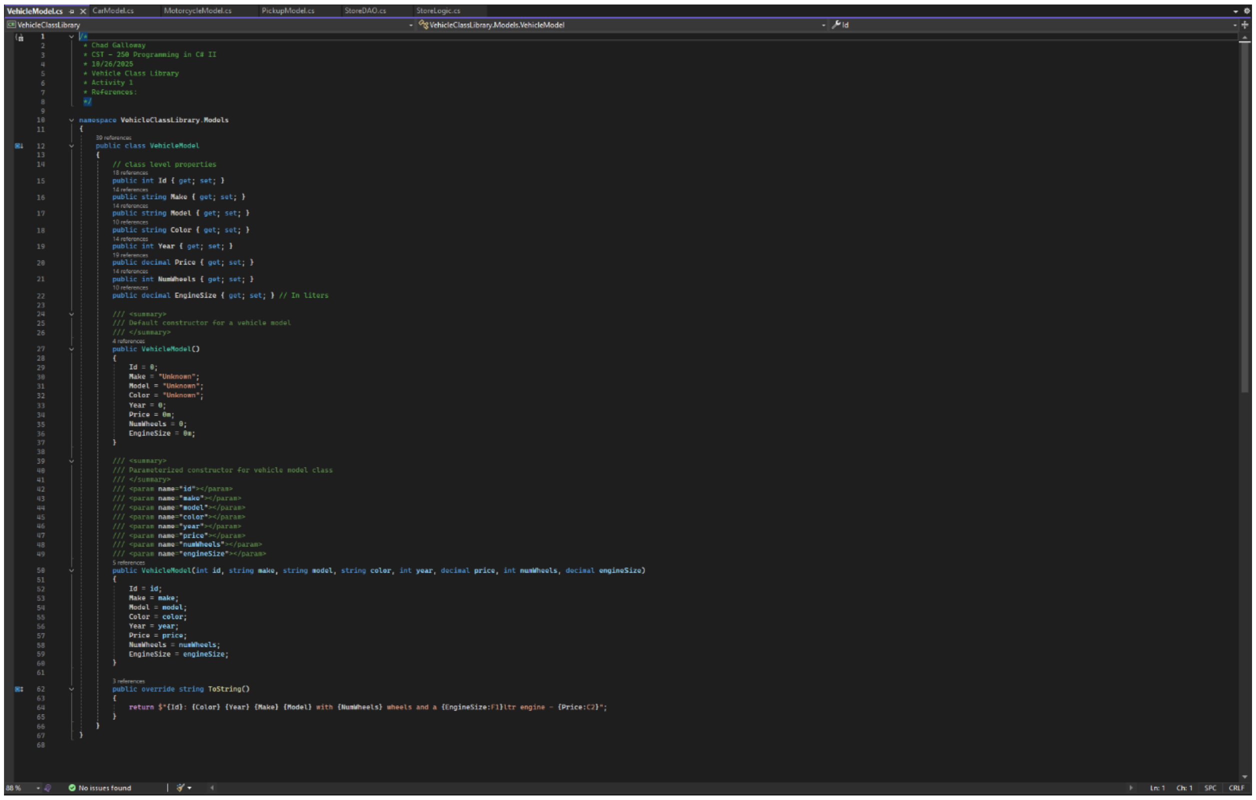
Figure 3: Screenshot of xUnit test



```
Test Explorer
5  5  0                                                        Search (Ctrl+I)
Test run finished: 5 Tests (5 Passed, 0 Failed, 0 Skipped) run in 74 ms      0 Warnings   0 Errors
Test                                              Duration  Traits  Error Message
VehicleClassLibrary.Tests (5)                     8 ms
  VehicleClassLibrary.Tests (5)                   8 ms
    StoreLogicTests (5)                           8 ms
      AddVehicleToCart_ShouldAddVehicle_WhenValidVehi...   1 ms
      AddVehicleToInventory_ShouldIncreaseInventoryCount   < 1 ms
      Checkout_ShouldReturnCorrectTotal_AndClearCart       7 ms
      GetInventory_ShouldReturnEmptyList_WhenNoVehicl...   < 1 ms
      GetShoppingCart_ShouldReturnEmptyList_WhenNoV...     < 1 ms

Run | Debug
Group Summary
  VehicleClassLibrary.Tests
    Tests in group: 5
    Total Duration: 8 ms
  Outcomes
    5 Passed
```

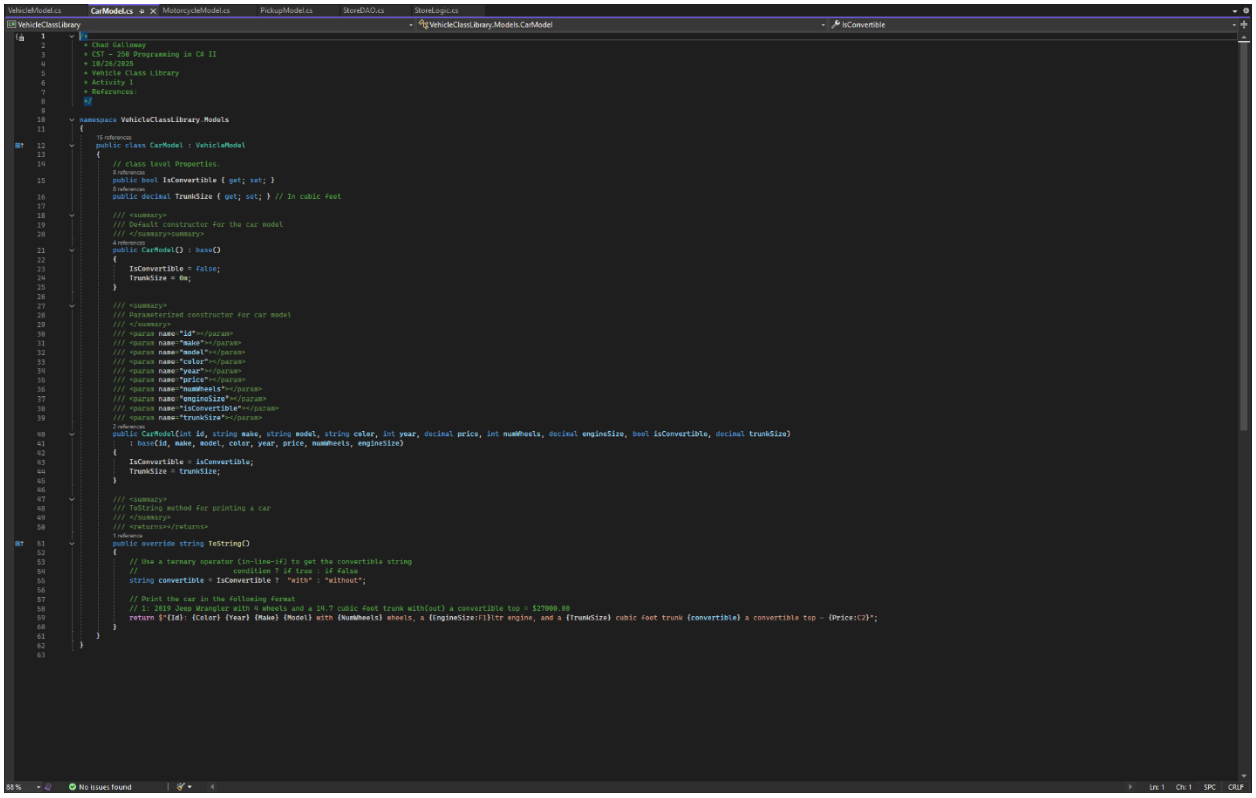
Figure 4: Screenshot of unit test

Figure 3 and 4 are screenshots of the unit tests. 3 is unsuccessful and 4 is successful completion of the tests.



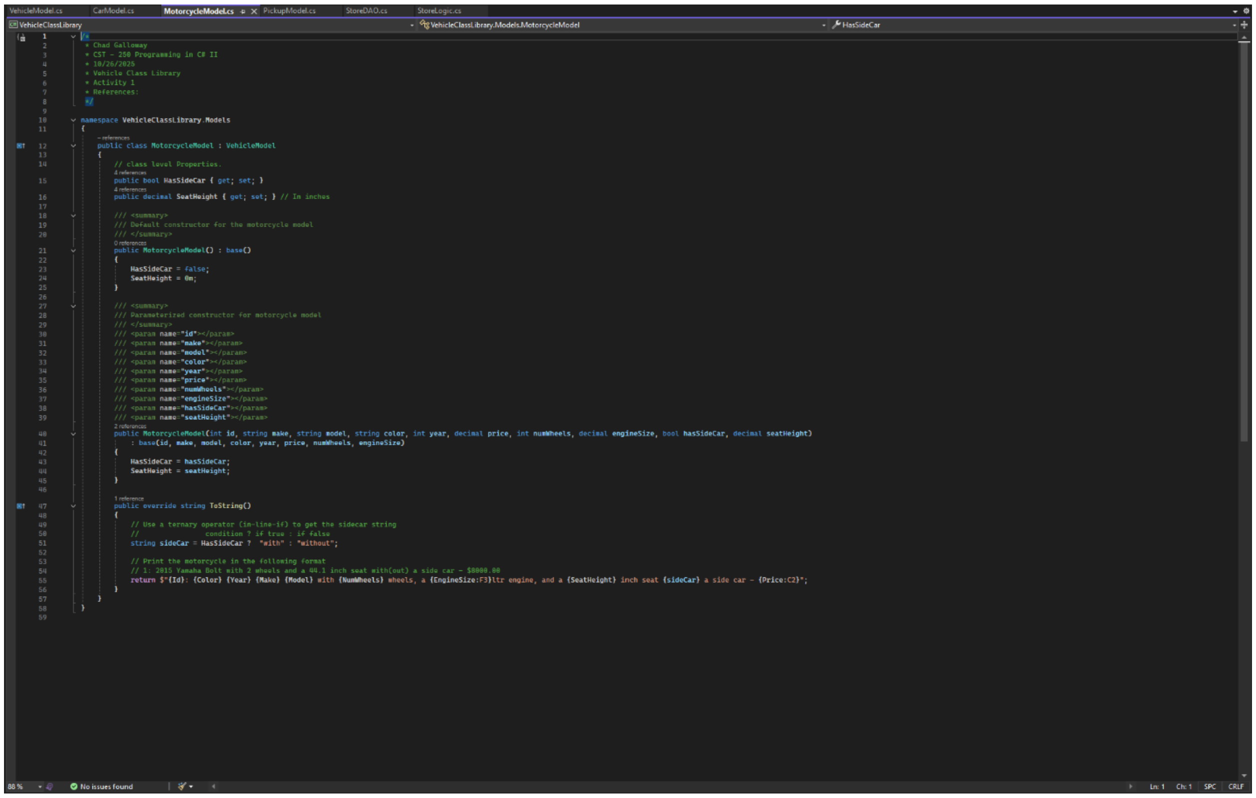
```csharp
/*
 * Chad Galloway
 * CST - 250 Programming in C# II
 * 10/26/2025
 * Vehicle Class Library
 * Activity 1
 * References:
 */

namespace VehicleClassLibrary.Models
{
    public class VehicleModel
    {
        // class level properties
        public int Id { get; set; }
        public string Make { get; set; }
        public string Model { get; set; }
        public string Color { get; set; }
        public int Year { get; set; }
        public decimal Price { get; set; }
        public int NumWheels { get; set; }
        public decimal EngineSize { get; set; } // In liters

        /// <summary>
        /// Default constructor for a vehicle model
        /// </summary>
        public VehicleModel()
        {
            Id = 0;
            Make = "Unknown";
            Model = "Unknown";
            Color = "Unknown";
            Year = 0;
            Price = 0m;
            NumWheels = 0;
            EngineSize = 0m;
        }

        /// <summary>
        /// Parameterized constructor for vehicle model class
        /// </summary>
        /// <param name="id"></param>
        /// <param name="make"></param>
        /// <param name="model"></param>
        /// <param name="color"></param>
        /// <param name="year"></param>
        /// <param name="price"></param>
        /// <param name="numWheels"></param>
        /// <param name="engineSize"></param>
        public VehicleModel(int id, string make, string model, string color, int year, decimal price, int numWheels, decimal engineSize)
        {
            Id = id;
            Make = make;
            Model = model;
            Color = color;
            Year = year;
            Price = price;
            NumWheels = numWheels;
            EngineSize = engineSize;
        }

        public override string ToString()
        {
            return $"{Id}: {Color} {Year} {Make} {Model} with {NumWheels} wheels and a {EngineSize:F1}ltr engine - {Price:C2}";
        }
    }
}
```

Figure 5: Vehicle Model



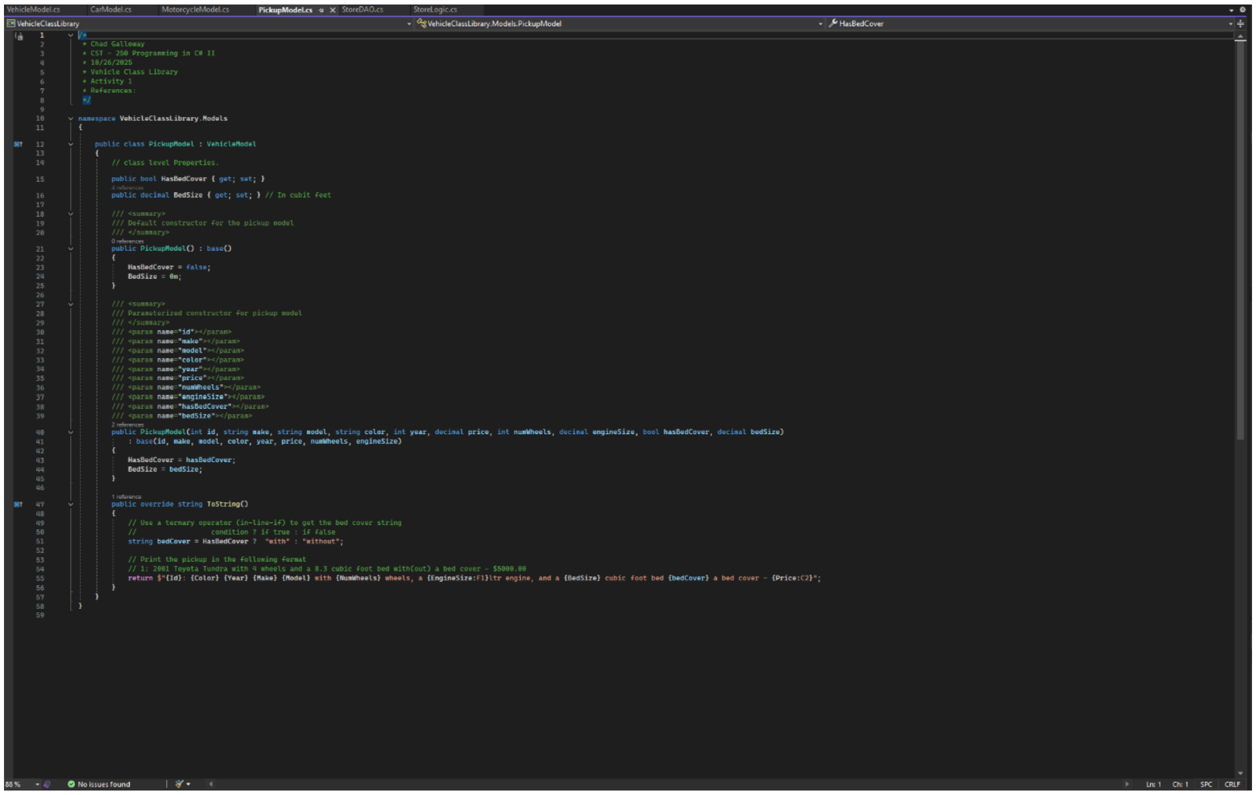
```csharp
/*
 * Chad Galloway
 * CST - 250 Programming in C# II
 * 10/26/2025
 * Vehicle Class Library
 * Activity 1
 * References:
 */

namespace VehicleClassLibrary.Models
{
    public class CarModel : VehicleModel
    {
        // class level Properties.
        public bool IsConvertible { get; set; }
        public decimal TrunkSize { get; set; } // In cubic feet

        /// <summary>
        /// Default constructor for the car model
        /// </summary>summary>
        public CarModel() : base()
        {
            IsConvertible = false;
            TrunkSize = 0m;
        }

        /// <summary>
        /// Parameterized constructor for car model
        /// </summary>
        /// <param name="id"></param>
        /// <param name="make"></param>
        /// <param name="model"></param>
        /// <param name="color"></param>
        /// <param name="year"></param>
        /// <param name="price"></param>
        /// <param name="numWheels"></param>
        /// <param name="engineSize"></param>
        /// <param name="isConvertible"></param>
        /// <param name="trunkSize"></param>
        public CarModel(int id, string make, string model, string color, int year, decimal price, int numWheels, decimal engineSize, bool isConvertible, decimal trunkSize)
            : base(id, make, model, color, year, price, numWheels, engineSize)
        {
            IsConvertible = isConvertible;
            TrunkSize = trunkSize;
        }

        /// <summary>
        /// ToString method for printing a car
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            // Use a ternary operator (in-line-if) to get the convertible string
            //                        condition ? if true : if false
            string convertible = IsConvertible ?  "with" : "without";

            // Print the car in the following format
            // 1: 2019 Jeep Wrangler with 4 wheels and a 14.7 cubic foot trunk with(out) a convertible top = $27000.00
            return $"{Id}: {Color} {Year} {Make} {Model} with {NumWheels} wheels, a {EngineSize:F1}ltr engine, and a {TrunkSize} cubic foot trunk {convertible} a convertible top - {Price:C2}";
        }
    }
}
```

Figure 6: Car Model



```
/*
 * Chad Galloway
 * CST - 250 Programming in C# II
 * 10/26/2025
 * Vehicle Class Library
 * Activity 1
 * References:
 */

namespace VehicleClassLibrary.Models
{
    public class MotorcycleModel : VehicleModel
    {
        // class level Properties.
        public bool HasSideCar { get; set; }
        public decimal SeatHeight { get; set; } // In inches

        /// <summary>
        /// Default constructor for the motorcycle model
        /// </summary>
        public MotorcycleModel() : base()
        {
            HasSideCar = false;
            SeatHeight = 0m;
        }

        /// <summary>
        /// Parameterized constructor for motorcycle model
        /// </summary>
        /// <param name="id"></param>
        /// <param name="make"></param>
        /// <param name="model"></param>
        /// <param name="color"></param>
        /// <param name="year"></param>
        /// <param name="price"></param>
        /// <param name="numWheels"></param>
        /// <param name="engineSize"></param>
        /// <param name="hasSideCar"></param>
        /// <param name="seatHeight"></param>
        public MotorcycleModel(int id, string make, string model, string color, int year, decimal price, int numWheels, decimal engineSize, bool hasSideCar, decimal seatHeight)
            : base(id, make, model, color, year, price, numWheels, engineSize)
        {
            HasSideCar = hasSideCar;
            SeatHeight = seatHeight;
        }

        public override string ToString()
        {
            // Use a ternary operator (in-line-if) to get the sidecar string
            //               condition ? if true : if false
            string sideCar = HasSideCar ?  "with" : "without";

            // Print the motorcycle in the following format
            // 1: 2015 Yamaha Bolt with 2 wheels and a 44.1 inch seat with(out) a side car - $8000.00
            return $"{Id}: {Color} {Year} {Make} {Model} with {NumWheels} wheels, a {EngineSize:F3}ltr engine, and a {SeatHeight} inch seat {sideCar} a side car - {Price:C2}";
        }
    }
}
```

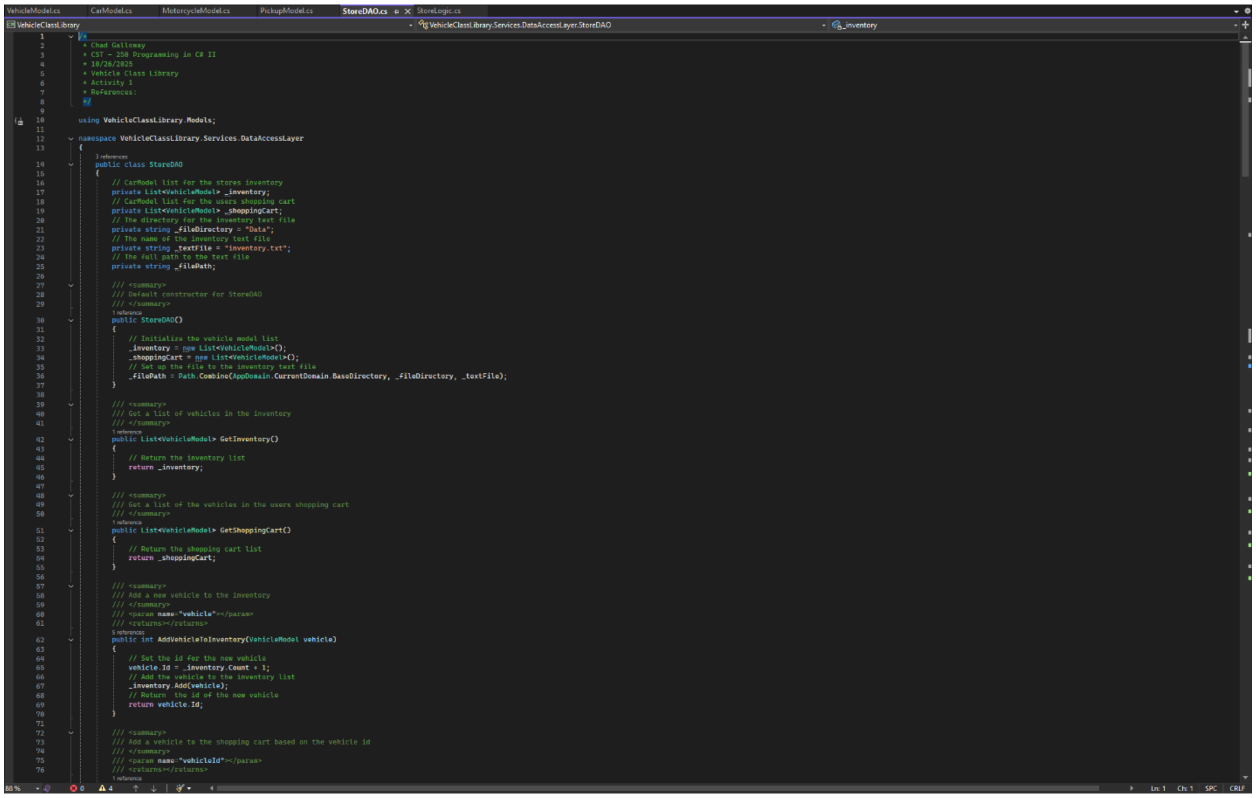
Figure 7: Motorcycle Model



```
/*
 * Chad Galloway
 * CST - 250 Programming in C# II
 * 10/26/2025
 * Vehicle Class Library
 * Activity 1
 * References:
 */

namespace VehicleClassLibrary.Models
{
    public class PickupModel : VehicleModel
    {
        // class level Properties.
        public bool HasBedCover { get; set; }
        public decimal BedSize { get; set; } // In cubit feet

        /// <summary>
        /// Default constructor for the pickup model
        /// </summary>
        public PickupModel() : base()
        {
            HasBedCover = false;
            BedSize = 0m;
        }

        /// <summary>
        /// Parameterized constructor for pickup model
        /// </summary>
        /// <param name="id"></param>
        /// <param name="make"></param>
        /// <param name="model"></param>
        /// <param name="color"></param>
        /// <param name="year"></param>
        /// <param name="price"></param>
        /// <param name="numWheels"></param>
        /// <param name="engineSize"></param>
        /// <param name="hasBedCover"></param>
        /// <param name="bedSize"></param>
        public PickupModel(int id, string make, string model, string color, int year, decimal price, int numWheels, decimal engineSize, bool hasBedCover, decimal bedSize)
            : base(id, make, model, color, year, price, numWheels, engineSize)
        {
            HasBedCover = hasBedCover;
            BedSize = bedSize;
        }

        public override string ToString()
        {
            // Use a ternary operator (in-line-if) to get the bed cover string
            //                condition ? if true : if false
            string bedCover = HasBedCover ?  "with" : "without";

            // Print the pickup in the following format
            // 1: 2001 Toyota Tundra with 4 wheels and a 8.3 cubic foot bed with(out) a bed cover - $5000.00
            return $"{Id}: {Color} {Year} {Make} {Model} with {NumWheels} wheels, a {EngineSize:F1}ltr engine, and a {BedSize} cubic foot bed {bedCover} a bed cover - {Price:C2}";
        }
    }
}
```

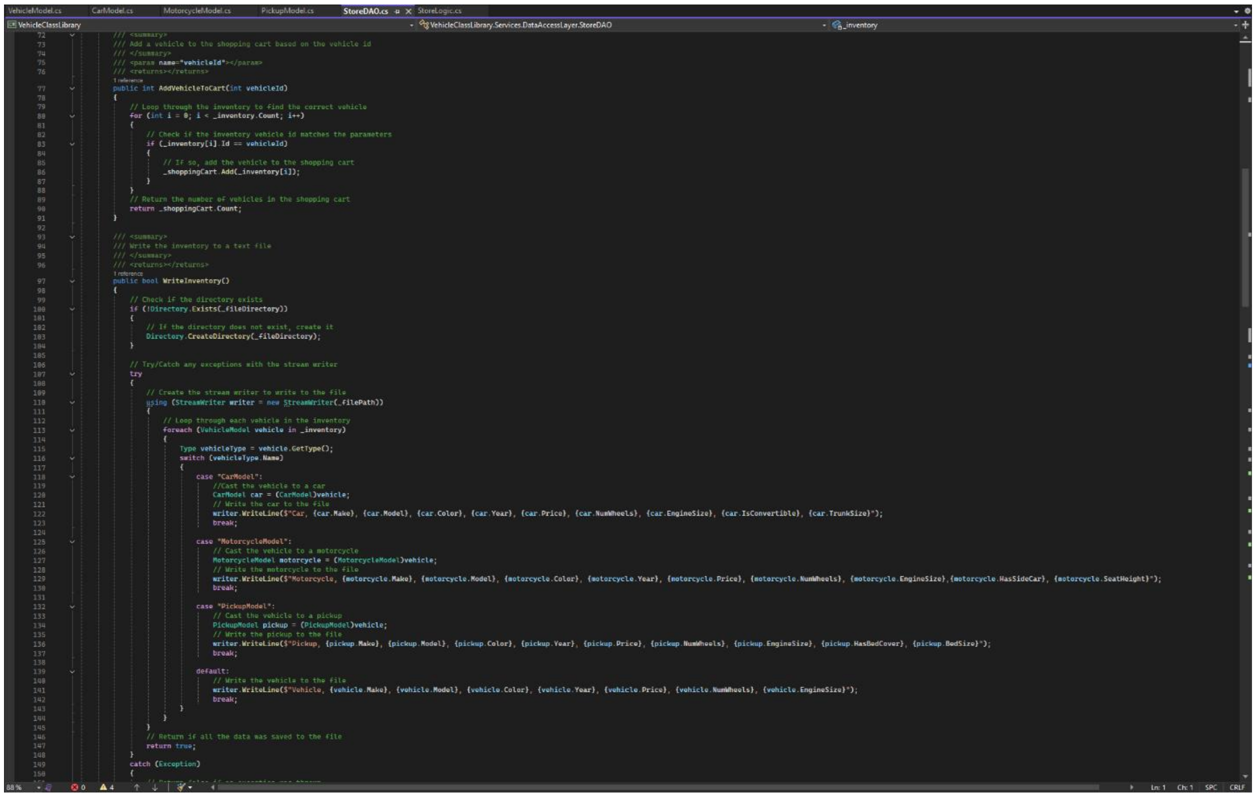
Figure 8: pickup model

Figures 5, 6, 7, and 8 are screen shots of the different models for vehicles in the application. Each screenshot covers the citations, constructors, and the overridden ToString() methods.



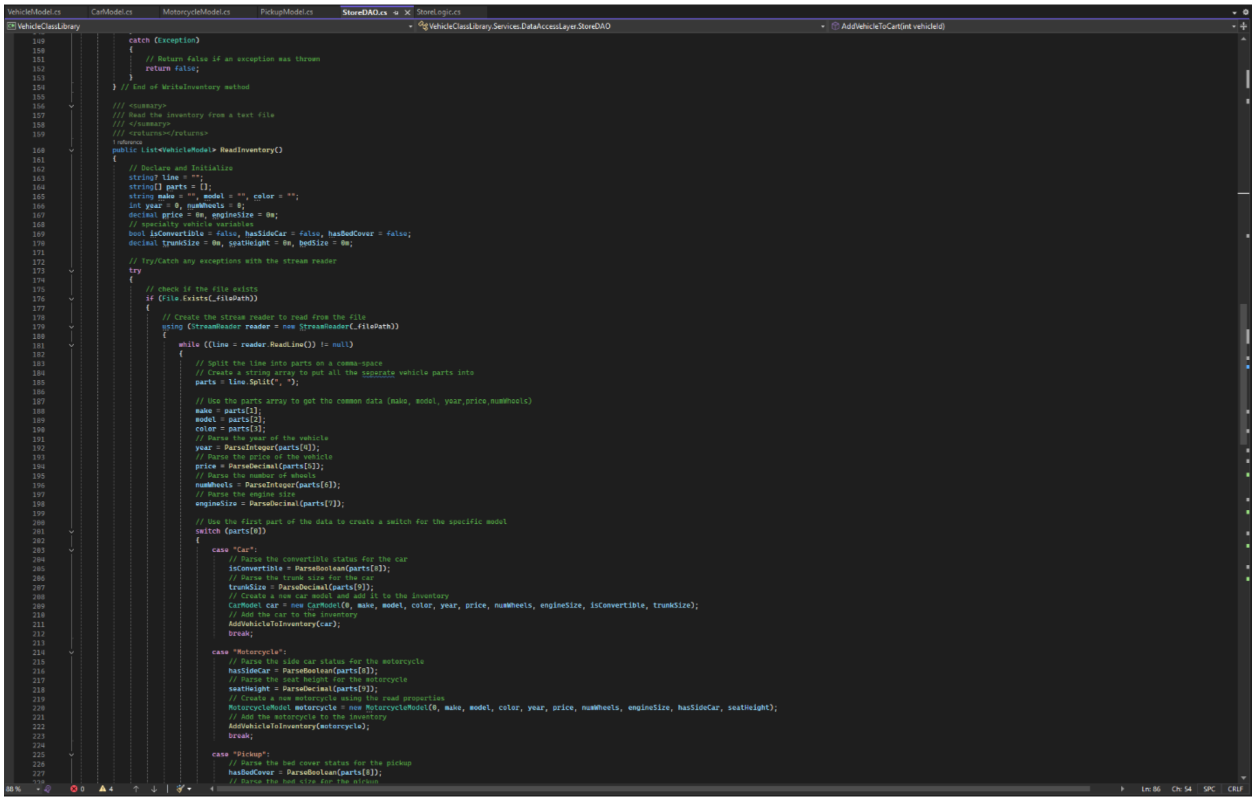
```
/*
 * Chad Galloway
 * CST - 250 Programming in C# II
 * 10/26/2025
 * Vehicle Class Library
 * Activity 1
 * References:
 */

using VehicleClassLibrary.Models;

namespace VehicleClassLibrary.Services.DataAccessLayer
{
    3 references
    public class StoreDAO
    {
        // CarModel list for the stores inventory
        private List<VehicleModel> _inventory;
        // CarModel list for the users shopping cart
        private List<VehicleModel> _shoppingCart;
        // The directory for the inventory text file
        private string _fileDirectory = "Data";
        // The name of the inventory text file
        private string _textFile = "inventory.txt";
        // The full path to the text file
        private string _filePath;

        /// <summary>
        /// Default constructor for StoreDAO
        /// </summary>
        1 reference
        public StoreDAO()
        {
            // Initialize the vehicle model list
            _inventory = new List<VehicleModel>();
            _shoppingCart = new List<VehicleModel>();
            // Set up the file to the inventory text file
            _filePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, _fileDirectory, _textFile);
        }

        /// <summary>
        /// Get a list of vehicles in the inventory
        /// </summary>
        1 reference
        public List<VehicleModel> GetInventory()
        {
            // Return the inventory list
            return _inventory;
        }

        /// <summary>
        /// Get a list of the vehicles in the users shopping cart
        /// </summary>
        1 reference
        public List<VehicleModel> GetShoppingCart()
        {
            // Return the shopping cart list
            return _shoppingCart;
        }

        /// <summary>
        /// Add a new vehicle to the inventory
        /// </summary>
        /// <param name="vehicle"></param>
        /// <returns></returns>
        5 references
        public int AddVehicleToInventory(VehicleModel vehicle)
        {
            // Set the id for the new vehicle
            vehicle.Id = _inventory.Count + 1;
            // Add the vehicle to the inventory list
            _inventory.Add(vehicle);
            // Return  the id of the new vehicle
            return vehicle.Id;
        }

        /// <summary>
        /// Add a vehicle to the shopping cart based on the vehicle id
        /// </summary>
        /// <param name="vehicleId"></param>
        /// <returns></returns>
        1 reference
```

Figure 9: StoreDAO citations and constructors.



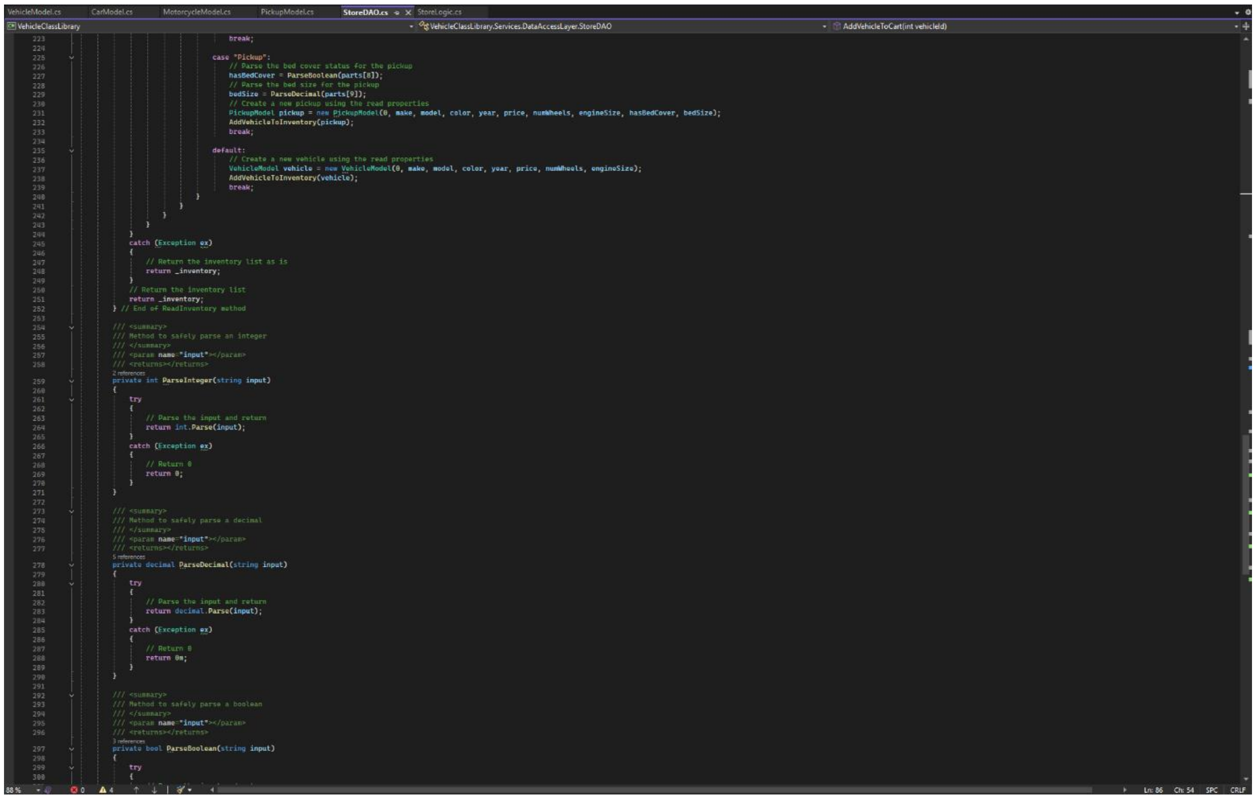
```csharp
/// <summary>
/// Add a vehicle to the shopping cart based on the vehicle id
/// </summary>
/// <param name="vehicleId"></param>
/// <returns></returns>
public int AddVehicleToCart(int vehicleId)
{
    // Loop through the inventory to find the correct vehicle
    for (int i = 0; i < _inventory.Count; i++)
    {
        // Check if the inventory vehicle id matches the parameters
        if (_inventory[i].Id == vehicleId)
        {
            // If so, add the vehicle to the shopping cart
            _shoppingCart.Add(_inventory[i]);
        }
    }
    // Return the number of vehicles in the shopping cart
    return _shoppingCart.Count;
}

/// <summary>
/// Write the inventory to a text file
/// </summary>
/// <returns></returns>
public bool WriteInventory()
{
    // Check if the directory exists
    if (!Directory.Exists(_fileDirectory))
    {
        // If the directory does not exist, create it
        Directory.CreateDirectory(_fileDirectory);
    }

    // Try/Catch any exceptions with the stream writer
    try
    {
        // Create the stream writer to write to the file
        using (StreamWriter writer = new StreamWriter(_filePath))
        {
            // Loop through each vehicle in the inventory
            foreach (VehicleModel vehicle in _inventory)
            {
                Type vehicleType = vehicle.GetType();
                switch (vehicleType.Name)
                {
                    case "CarModel":
                        //Cast the vehicle to a car
                        CarModel car = (CarModel)vehicle;
                        // Write the car to the file
                        writer.WriteLine($"Car, {car.Make}, {car.Model}, {car.Color}, {car.Year}, {car.Price}, {car.NumWheels}, {car.EngineSize}, {car.IsConvertible}, {car.TrunkSize}");
                        break;

                    case "MotorcycleModel":
                        // Cast the vehicle to a motorcycle
                        MotorcycleModel motorcycle = (MotorcycleModel)vehicle;
                        // Write the motorcycle to the file
                        writer.WriteLine($"Motorcycle, {motorcycle.Make}, {motorcycle.Model}, {motorcycle.Color}, {motorcycle.Year}, {motorcycle.Price}, {motorcycle.NumWheels}, {motorcycle.EngineSize},{motorcycle.HasSideCar}, {motorcycle.SeatHeight}");
                        break;

                    case "PickupModel":
                        // Cast the vehicle to a pickup
                        PickupModel pickup = (PickupModel)vehicle;
                        // Write the pickup to the file
                        writer.WriteLine($"Pickup, {pickup.Make}, {pickup.Model}, {pickup.Color}, {pickup.Year}, {pickup.Price}, {pickup.NumWheels}, {pickup.EngineSize}, {pickup.HasBedCover}, {pickup.BedSize}");
                        break;

                    default:
                        // Write the vehicle to the file
                        writer.WriteLine($"Vehicle, {vehicle.Make}, {vehicle.Model}, {vehicle.Color}, {vehicle.Year}, {vehicle.Price}, {vehicle.NumWheels}, {vehicle.EngineSize}");
                        break;
                }
            }
        }
        // Return if all the data was saved to the file
        return true;
    }
    catch (Exception)
    {
```

Figure 10: continuation of StoreDAO



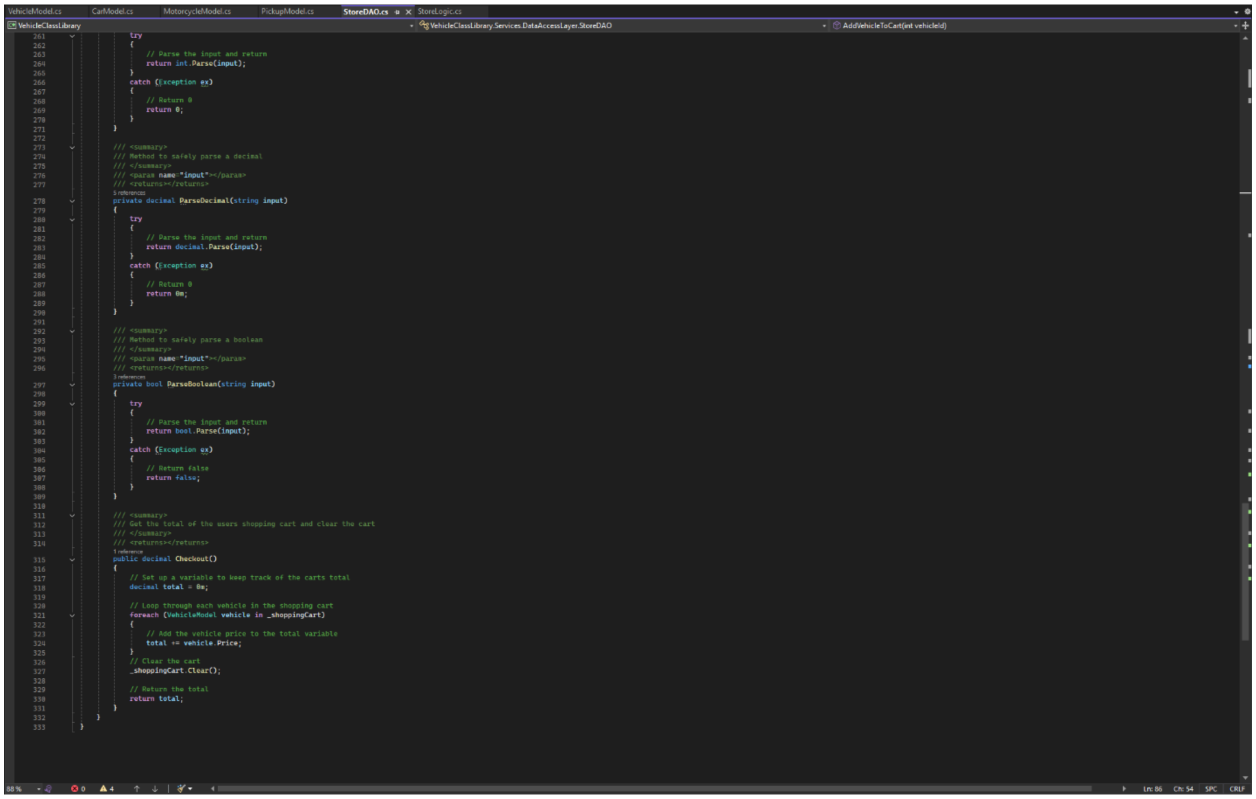
```csharp
    catch (Exception)
    {
        // Return false if an exception was thrown
        return false;
    }
} // End of WriteInventory method

/// <summary>
/// Read the inventory from a text file
/// </summary>
/// <returns></returns>
public List<VehicleModel> ReadInventory()
{
    // Declare and Initialize
    string? line = "";
    string[] parts = [];
    string make = "", model = "", color = "";
    int year = 0, numWheels = 0;
    decimal price = 0m, engineSize = 0m;
    // specialty vehicle variables
    bool isConvertible = false, hasSideCar = false, hasBedCover = false;
    decimal trunkSize = 0m, seatHeight = 0m, bedSize = 0m;

    // Try/Catch any exceptions with the stream reader
    try
    {
        // check if the file exists
        if (File.Exists(_filePath))
        {
            // Create the stream reader to read from the file
            using (StreamReader reader = new StreamReader(_filePath))
            {
                while ((line = reader.ReadLine()) != null)
                {
                    // Split the line into parts on a comma-space
                    // Create a string array to put all the seperate vehicle parts into
                    parts = line.Split(", ");

                    // Use the parts array to get the common data (make, model, year,price,numWheels)
                    make = parts[1];
                    model = parts[2];
                    color = parts[3];
                    // Parse the year of the vehicle
                    year = ParseInteger(parts[4]);
                    // Parse the price of the vehicle
                    price = ParseDecimal(parts[5]);
                    // Parse the number of wheels
                    numWheels = ParseInteger(parts[6]);
                    // Parse the engine size
                    engineSize = ParseDecimal(parts[7]);

                    // Use the first part of the data to create a switch for the specific model
                    switch (parts[0])
                    {
                        case "Car":
                            // Parse the convertible status for the car
                            isConvertible = ParseBoolean(parts[8]);
                            // Parse the trunk size for the car
                            trunkSize = ParseDecimal(parts[9]);
                            // Create a new car model and add it to the inventory
                            CarModel car = new CarModel(0, make, model, color, year, price, numWheels, engineSize, isConvertible, trunkSize);
                            // Add the car to the inventory
                            AddVehicleToInventory(car);
                            break;

                        case "Motorcycle":
                            // Parse the side car status for the motorcycle
                            hasSideCar = ParseBoolean(parts[8]);
                            // Parse the seat height for the motorcycle
                            seatHeight = ParseDecimal(parts[9]);
                            // Create a new motorcycle using the read properties
                            MotorcycleModel motorcycle = new MotorcycleModel(0, make, model, color, year, price, numWheels, engineSize, hasSideCar, seatHeight);
                            // Add the motorcycle to the inventory
                            AddVehicleToInventory(motorcycle);
                            break;

                        case "Pickup":
                            // Parse the bed cover status for the pickup
                            hasBedCover = ParseBoolean(parts[8]);
                            // Parse the bed size for the pickup
```

Figure 11: continuation of StoreDAO



```csharp
                                    break;

                                case "Pickup":
                                    // Parse the bed cover status for the pickup
                                    hasBedCover = ParseBoolean(parts[8]);
                                    // Parse the bed size for the pickup
                                    bedSize = ParseDecimal(parts[9]);
                                    // Create a new pickup using the read properties
                                    PickupModel pickup = new PickupModel(0, make, model, color, year, price, numWheels, engineSize, hasBedCover, bedSize);
                                    AddVehicleToInventory(pickup);
                                    break;

                                default:
                                    // Create a new vehicle using the read properties
                                    VehicleModel vehicle = new VehicleModel(0, make, model, color, year, price, numWheels, engineSize);
                                    AddVehicleToInventory(vehicle);
                                    break;
                            }
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                // Return the inventory list as is
                return _inventory;
            }
            // Return the inventory list
            return _inventory;
        } // End of ReadInventory method

        /// <summary>
        /// Method to safely parse an integer
        /// </summary>
        /// <param name="input"></param>
        /// <returns></returns>
        private int ParseInteger(string input)
        {
            try
            {
                // Parse the input and return
                return int.Parse(input);
            }
            catch (Exception ex)
            {
                // Return 0
                return 0;
            }
        }

        /// <summary>
        /// Method to safely parse a decimal
        /// </summary>
        /// <param name="input"></param>
        /// <returns></returns>
        private decimal ParseDecimal(string input)
        {
            try
            {
                // Parse the input and return
                return decimal.Parse(input);
            }
            catch (Exception ex)
            {
                // Return 0
                return 0m;
            }
        }

        /// <summary>
        /// Method to safely parse a boolean
        /// </summary>
        /// <param name="input"></param>
        /// <returns></returns>
        private bool ParseBoolean(string input)
        {
            try
            {
```

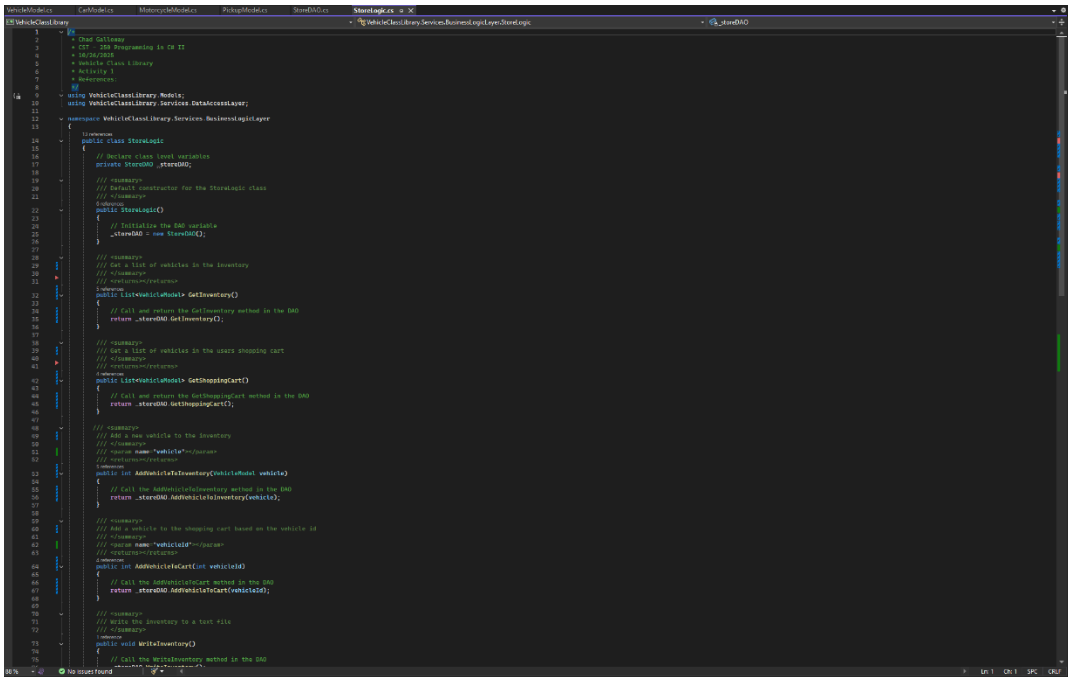
Figure 12: continuation of StoreDAO



```csharp
            try
            {
                // Parse the input and return
                return int.Parse(input);
            }
            catch (Exception ex)
            {
                // Return 0
                return 0;
            }
        }

        /// <summary>
        /// Method to safely parse a decimal
        /// </summary>
        /// <param name="input"></param>
        /// <returns></returns>
        private decimal ParseDecimal(string input)
        {
            try
            {
                // Parse the input and return
                return decimal.Parse(input);
            }
            catch (Exception ex)
            {
                // Return 0
                return 0m;
            }
        }

        /// <summary>
        /// Method to safely parse a boolean
        /// </summary>
        /// <param name="input"></param>
        /// <returns></returns>
        private bool ParseBoolean(string input)
        {
            try
            {
                // Parse the input and return
                return bool.Parse(input);
            }
            catch (Exception ex)
            {
                // Return false
                return false;
            }
        }

        /// <summary>
        /// Get the total of the users shopping cart and clear the cart
        /// </summary>
        /// <returns></returns>
        public decimal Checkout()
        {
            // Set up a variable to keep track of the carts total
            decimal total = 0m;

            // Loop through each vehicle in the shopping cart
            foreach (VehicleModel vehicle in _shoppingCart)
            {
                // Add the vehicle price to the total variable
                total += vehicle.Price;
            }
            // Clear the cart
            _shoppingCart.Clear();

            // Return the total
            return total;
        }
    }
}
```

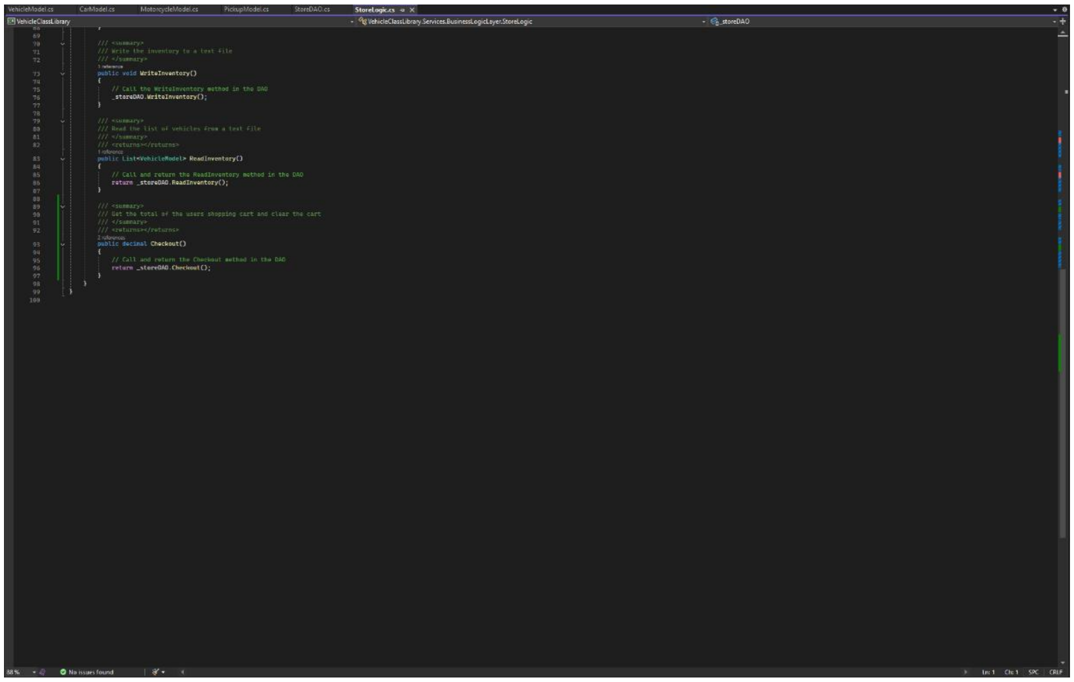
Figure 13: Final screenshot of StoreDAO

Figures 9-13 are screen shots of the StoreDAO class in its entirety. Every method is represented in the screen shots.



```
/*
 * Chad Galloway
 * CST - 250 Programming in C# II
 * 10/26/2025
 * Vehicle Class Library
 * Activity 1
 * References:
 */
using VehicleClassLibrary.Models;
using VehicleClassLibrary.Services.DataAccessLayer;

namespace VehicleClassLibrary.Services.BusinessLogicLayer
{
    public class StoreLogic
    {
        // Declare class level variables
        private StoreDAO _storeDAO;

        /// <summary>
        /// Default constructor for the StoreLogic class
        /// </summary>
        public StoreLogic()
        {
            // Initialize the DAO variable
            _storeDAO = new StoreDAO();
        }

        /// <summary>
        /// Get a list of vehicles in the inventory
        /// </summary>
        /// <returns></returns>
        public List<VehicleModel> GetInventory()
        {
            // Call and return the GetInventory method in the DAO
            return _storeDAO.GetInventory();
        }

        /// <summary>
        /// Get a list of vehicles in the users shopping cart
        /// </summary>
        /// <returns></returns>
        public List<VehicleModel> GetShoppingCart()
        {
            // Call and return the GetShoppingCart method in the DAO
            return _storeDAO.GetShoppingCart();
        }

       /// <summary>
        /// Add a new vehicle to the inventory
        /// </summary>
        /// <param name="vehicle"></param>
        /// <returns></returns>
        public int AddVehicleToInventory(VehicleModel vehicle)
        {
            // Call the AddVehicleToInventory method in the DAO
            return _storeDAO.AddVehicleToInventory(vehicle);
        }

        /// <summary>
        /// Add a vehicle to the shopping cart based on the vehicle id
        /// </summary>
        /// <param name="vehicleId"></param>
        /// <returns></returns>
        public int AddVehicleToCart(int vehicleId)
        {
            // Call the AddVehicleToCart method in the DAO
            return _storeDAO.AddVehicleToCart(vehicleId);
        }

        /// <summary>
        /// Write the inventory to a text file
        /// </summary>
        public void WriteInventory()
        {
            // Call the WriteInventory method in the DAO
```

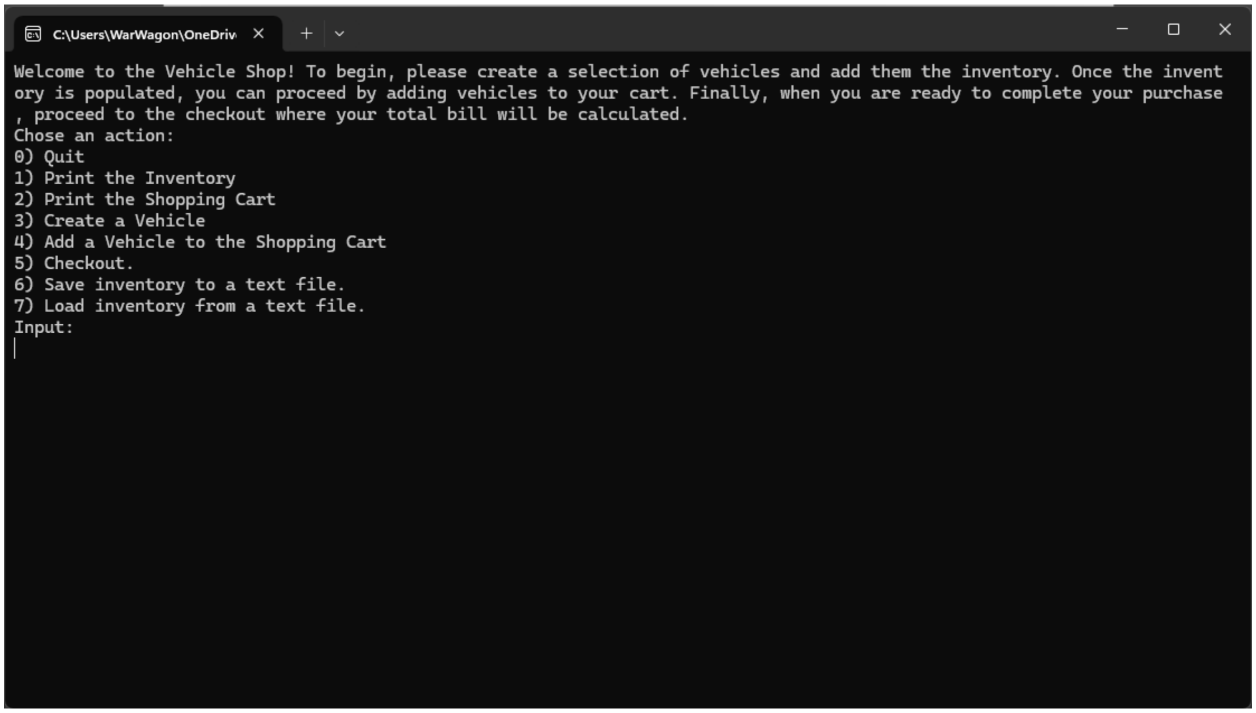
Figure 14: StoreLogic class citations and constructor



```
        /// <summary>
        /// Write the inventory to a text file
        /// </summary>
        public void WriteInventory()
        {
            // Call the WriteInventory method in the DAO
            _storeDAO.WriteInventory();
        }

        /// <summary>
        /// Read the list of vehicles from a text file
        /// </summary>
        /// <returns></returns>
        public List<VehicleModel> ReadInventory()
        {
            // Call and return the ReadInventory method in the DAO
            return _storeDAO.ReadInventory();
        }

        /// <summary>
        /// Get the total of the users shopping cart and clear the cart
        /// </summary>
        /// <returns></returns>
        public decimal Checkout()
        {
            // Call and return the Checkout method in the DAO
            return _storeDAO.Checkout();
        }
    }
}
```

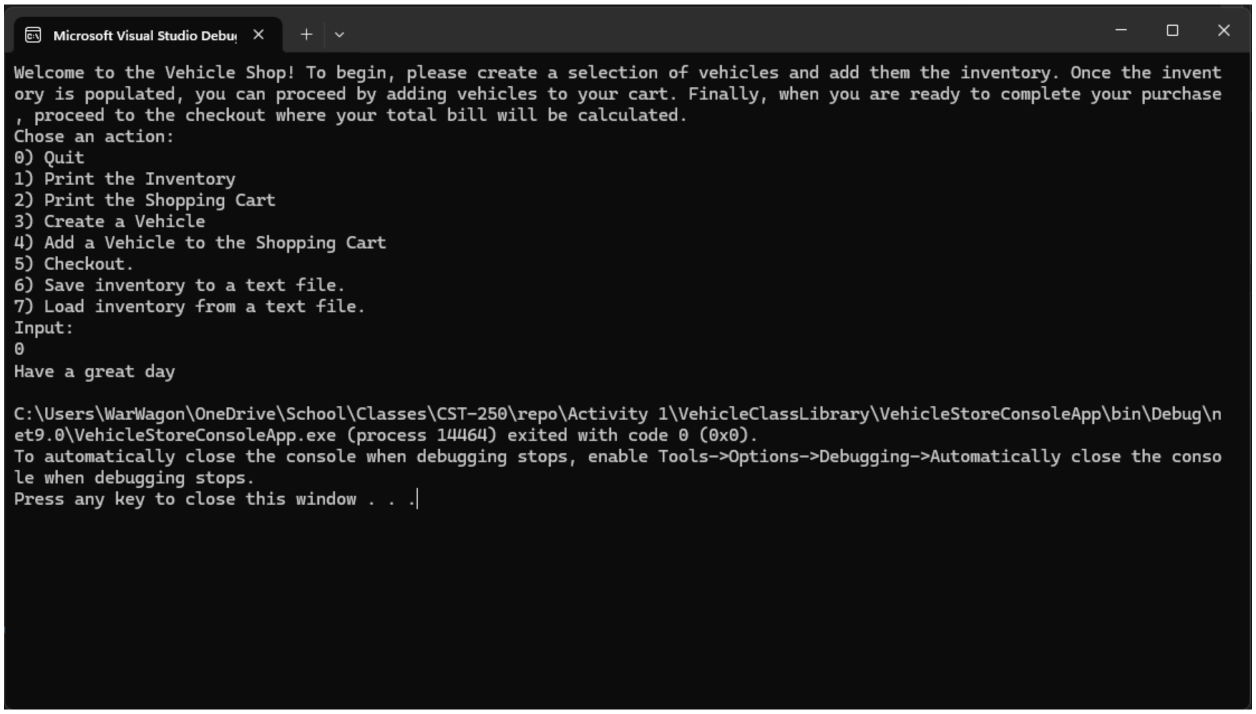
Figure 15: StoreLogic methods

Figure 14 and 15 are screenshots of the StoreLogic class.



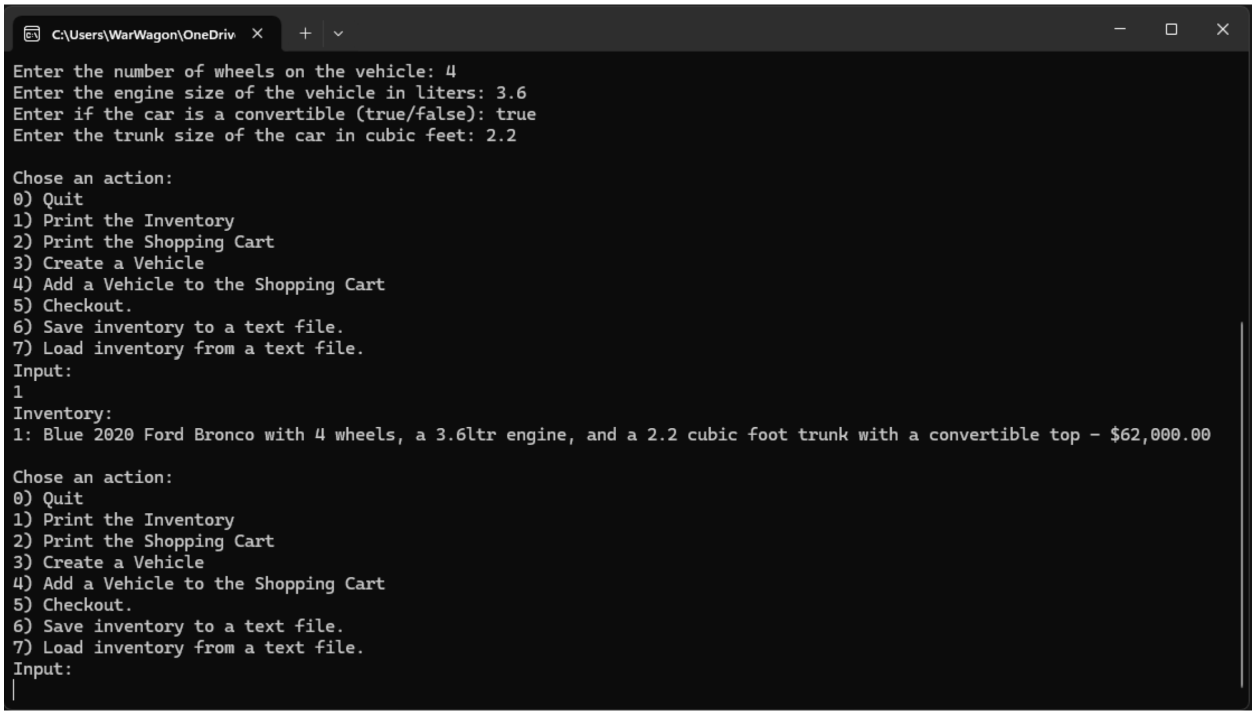
```
Welcome to the Vehicle Shop! To begin, please create a selection of vehicles and add them the inventory. Once the invent
ory is populated, you can proceed by adding vehicles to your cart. Finally, when you are ready to complete your purchase
, proceed to the checkout where your total bill will be calculated.
Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
```

Figure 16: Initial load state of the console app



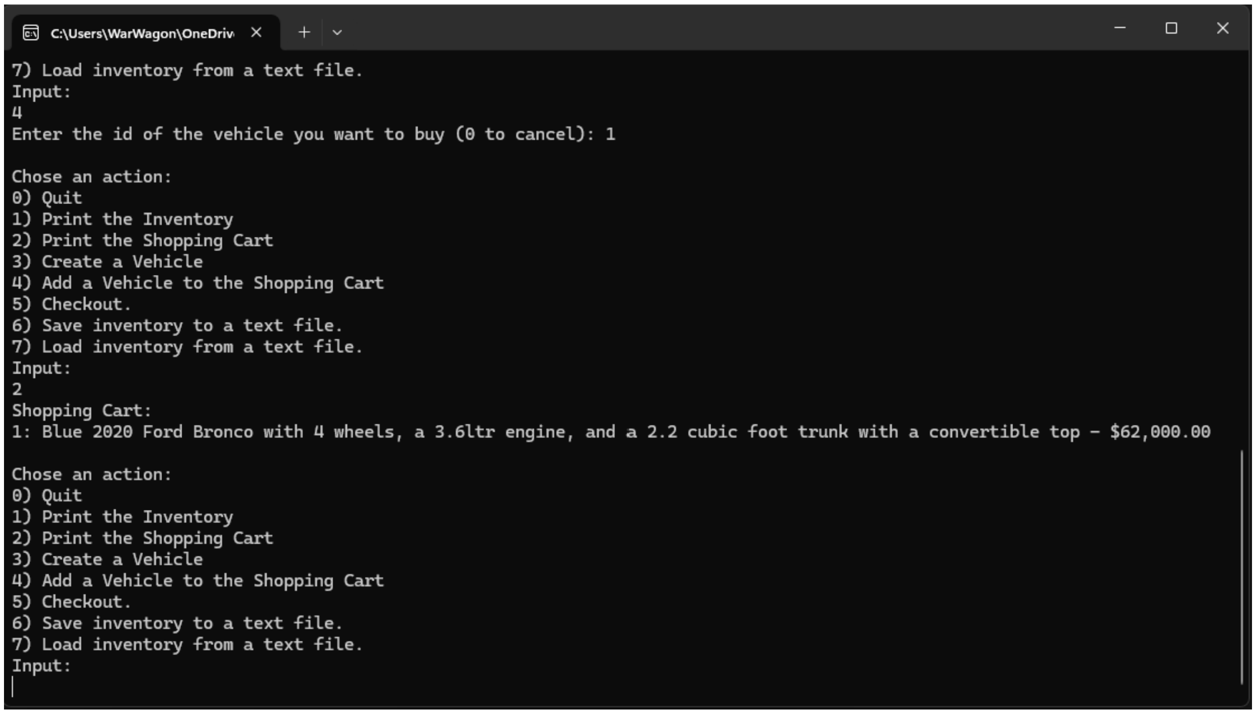
```
Welcome to the Vehicle Shop! To begin, please create a selection of vehicles and add them the inventory. Once the invent
ory is populated, you can proceed by adding vehicles to your cart. Finally, when you are ready to complete your purchase
, proceed to the checkout where your total bill will be calculated.
Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
0
Have a great day

C:\Users\WarWagon\OneDrive\School\Classes\CST-250\repo\Activity 1\VehicleClassLibrary\VehicleStoreConsoleApp\bin\Debug\n
et9.0\VehicleStoreConsoleApp.exe (process 14464) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Figure 17: user input 0



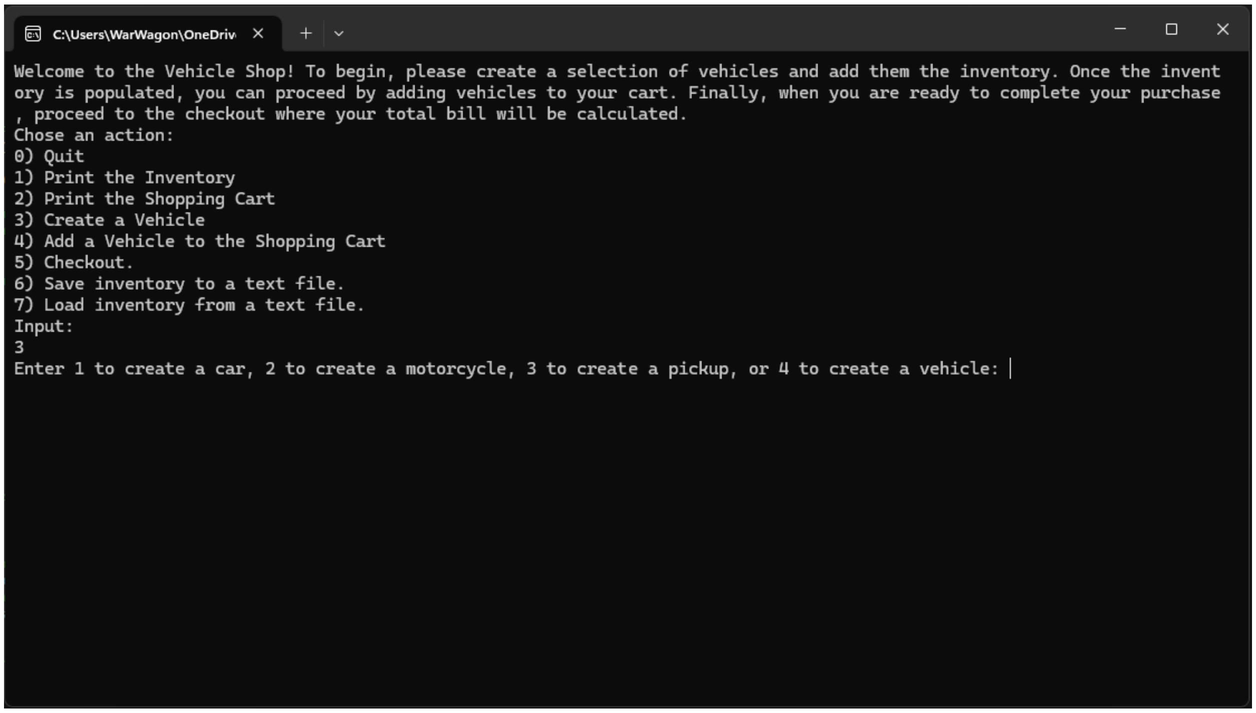
```
Enter the number of wheels on the vehicle: 4
Enter the engine size of the vehicle in liters: 3.6
Enter if the car is a convertible (true/false): true
Enter the trunk size of the car in cubic feet: 2.2

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
1
Inventory:
1: Blue 2020 Ford Bronco with 4 wheels, a 3.6ltr engine, and a 2.2 cubic foot trunk with a convertible top - $62,000.00

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
```

Figure 18: user input 1



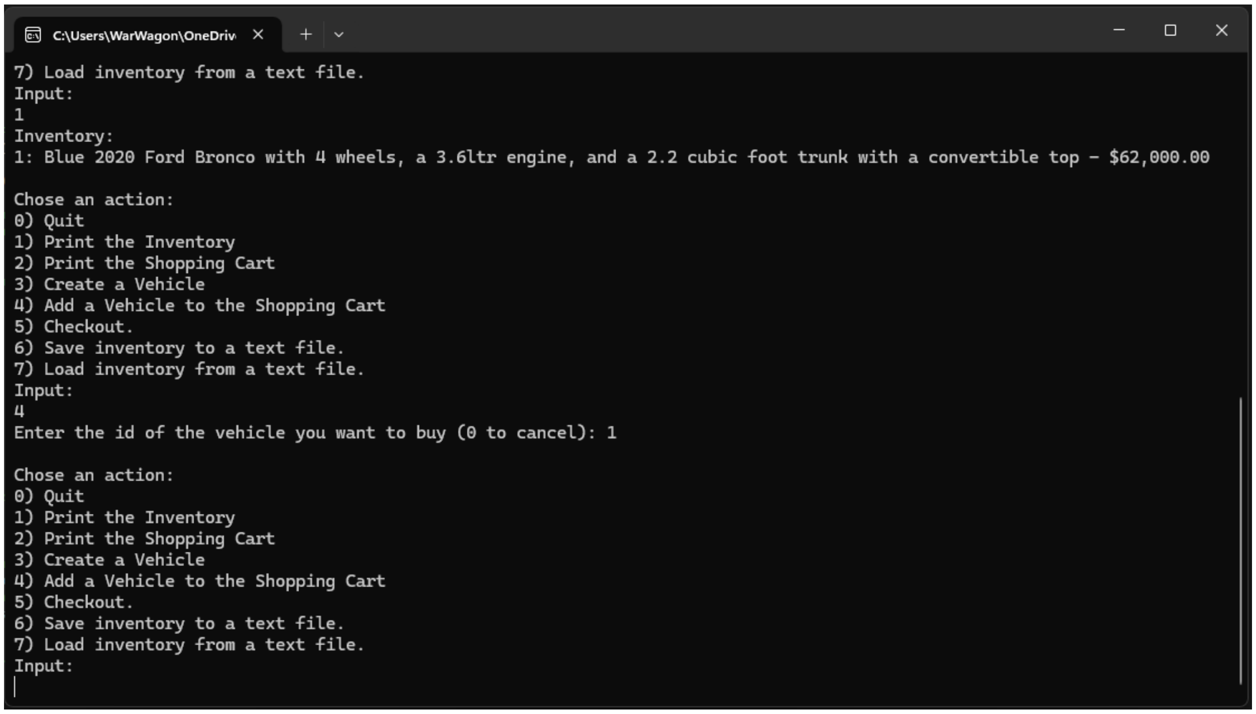
```
7) Load inventory from a text file.
Input:
4
Enter the id of the vehicle you want to buy (0 to cancel): 1

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
2
Shopping Cart:
1: Blue 2020 Ford Bronco with 4 wheels, a 3.6ltr engine, and a 2.2 cubic foot trunk with a convertible top - $62,000.00

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
```

Figure 19: user input 2



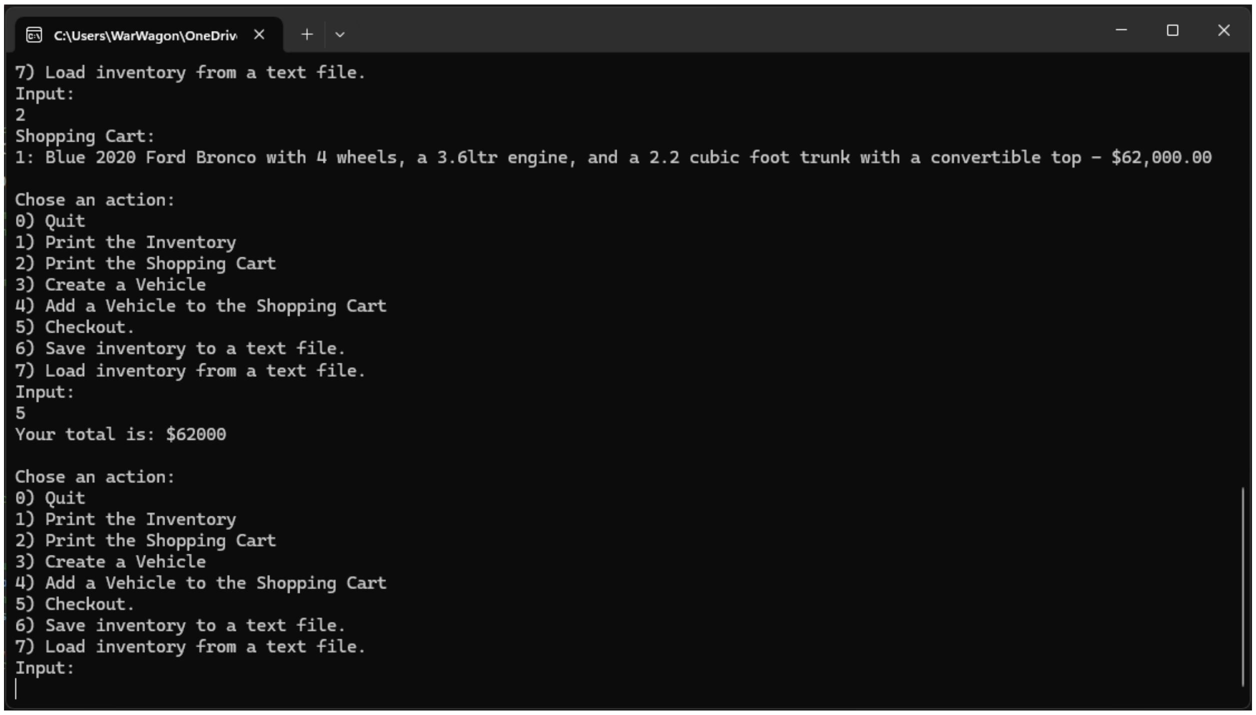
```
Welcome to the Vehicle Shop! To begin, please create a selection of vehicles and add them the inventory. Once the invent
ory is populated, you can proceed by adding vehicles to your cart. Finally, when you are ready to complete your purchase
, proceed to the checkout where your total bill will be calculated.
Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
3
Enter 1 to create a car, 2 to create a motorcycle, 3 to create a pickup, or 4 to create a vehicle:
```

Figure 20: user input 3



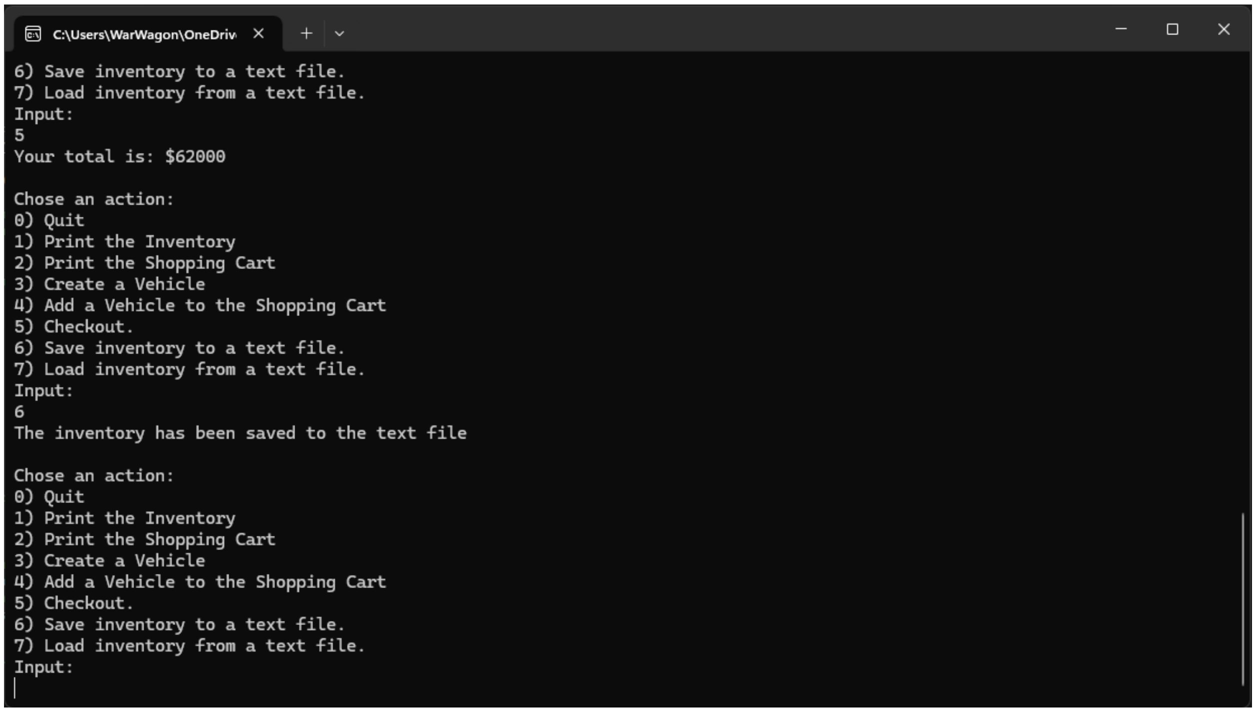
```
7) Load inventory from a text file.
Input:
1
Inventory:
1: Blue 2020 Ford Bronco with 4 wheels, a 3.6ltr engine, and a 2.2 cubic foot trunk with a convertible top - $62,000.00

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
4
Enter the id of the vehicle you want to buy (0 to cancel): 1

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
```

Figure 21: user input 4



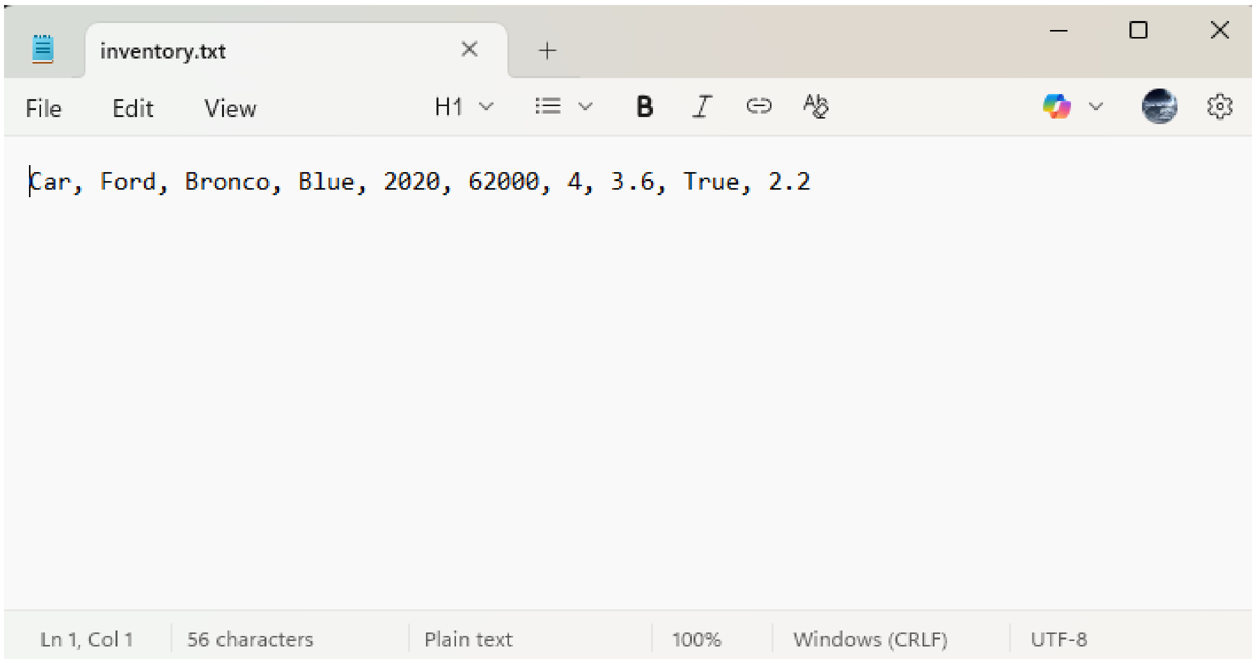
```
7) Load inventory from a text file.
Input:
2
Shopping Cart:
1: Blue 2020 Ford Bronco with 4 wheels, a 3.6ltr engine, and a 2.2 cubic foot trunk with a convertible top - $62,000.00

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
5
Your total is: $62000

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
```

Figure 22: user input 5



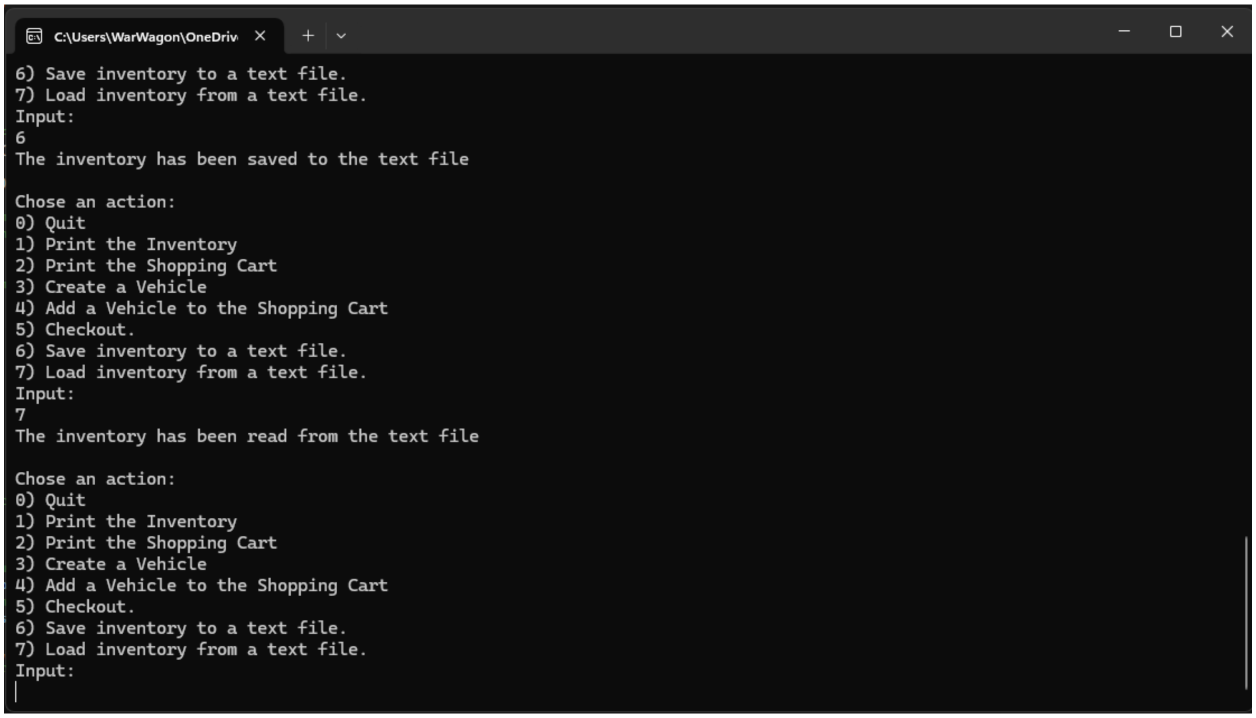
```
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
5
Your total is: $62000

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
6
The inventory has been saved to the text file

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
```

Figure 23: user input 6



```
Car, Ford, Bronco, Blue, 2020, 62000, 4, 3.6, True, 2.2
```

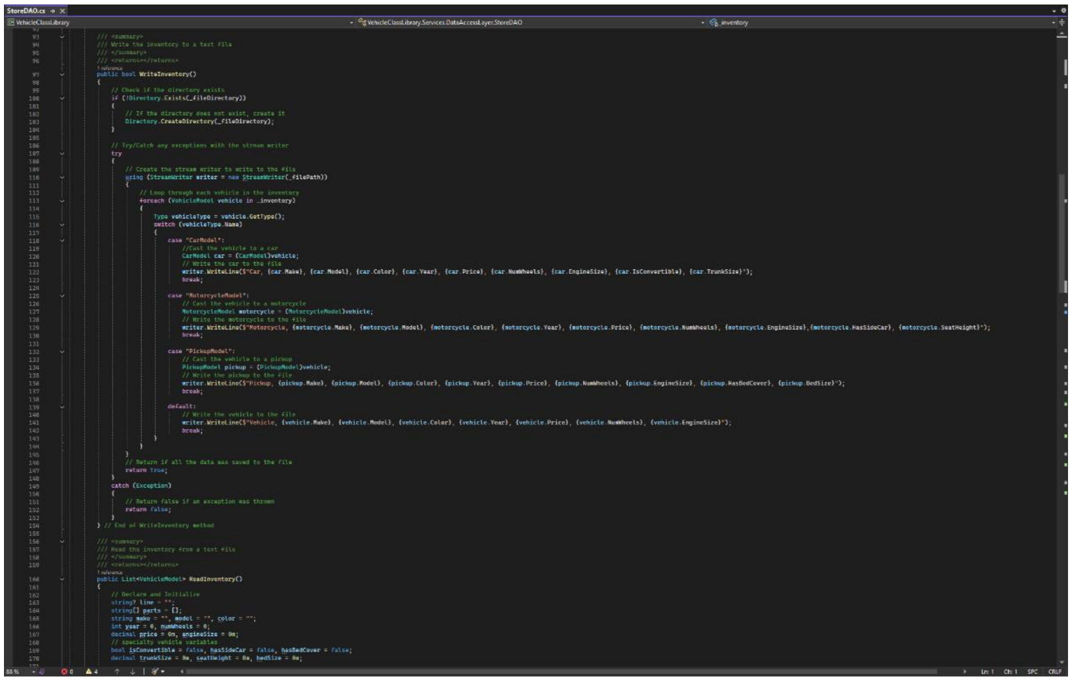
Figure 24: File contents after save (number 6)



```
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
6
The inventory has been saved to the text file

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
7
The inventory has been read from the text file

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
```

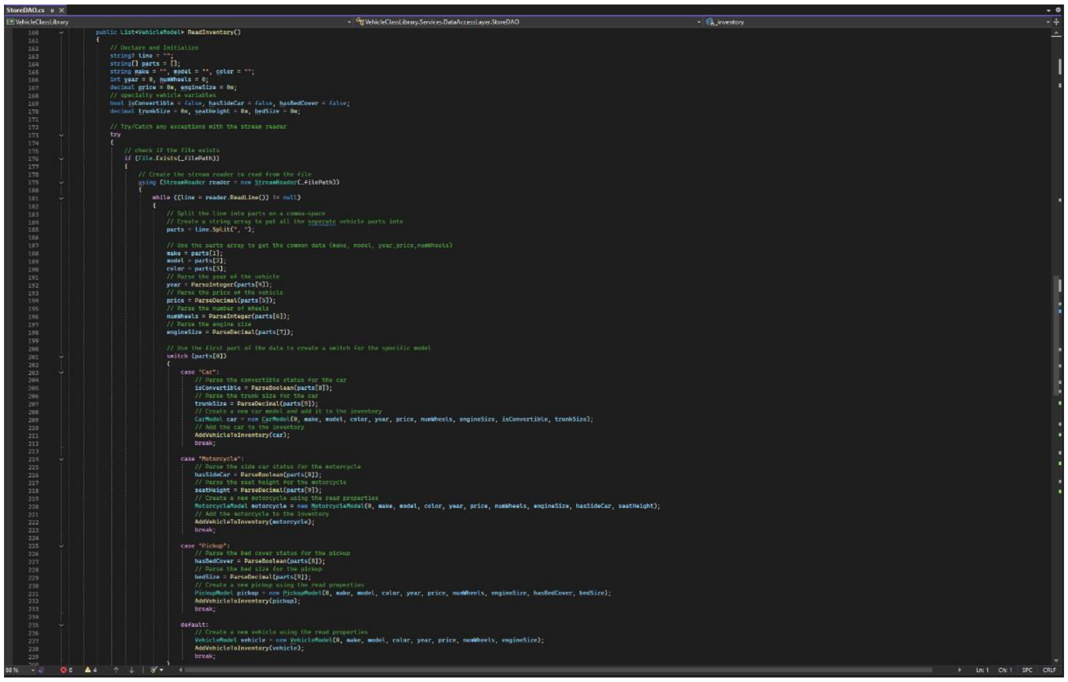
Figure 25: user input 7

Figure 16- 25 are screenshots of the application running and demonstrating the different menu option selections made by the user



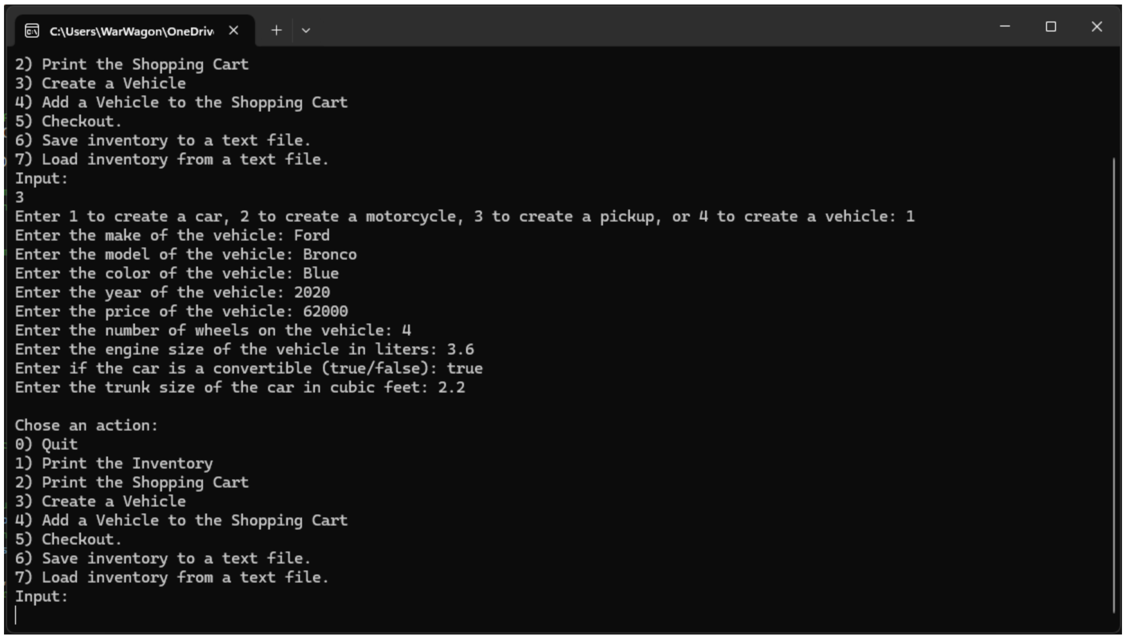
```csharp
        /// <summary>
        /// Write the inventory to a text file
        /// </summary>
        /// <returns></returns>
        public bool WriteInventory()
        {
            // Check if the directory exists
            if (!Directory.Exists(_fileDirectory))
            {
                // If the directory does not exist, create it
                Directory.CreateDirectory(_fileDirectory);
            }

            // Try/Catch any exceptions with the stream writer
            try
            {
                // Create the stream writer to write to the file
                using (StreamWriter writer = new StreamWriter(_filePath))
                {
                    // Loop through each vehicle in the inventory
                    foreach (VehicleModel vehicle in _inventory)
                    {
                        Type vehicleType = vehicle.GetType();
                        switch (vehicleType.Name)
                        {
                            case "CarModel":
                                //Cast the vehicle to a car
                                CarModel car = (CarModel)vehicle;
                                // Write the car to the file
                                writer.WriteLine($"Car, {car.Make}, {car.Model}, {car.Color}, {car.Year}, {car.Price}, {car.NumWheels}, {car.EngineSize}, {car.IsConvertible}, {car.TrunkSize}");
                                break;

                            case "MotorcycleModel":
                                // Cast the vehicle to a motorcycle
                                MotorcycleModel motorcycle = (MotorcycleModel)vehicle;
                                // Write the motorcycle to the file
                                writer.WriteLine($"Motorcycle, {motorcycle.Make}, {motorcycle.Model}, {motorcycle.Color}, {motorcycle.Year}, {motorcycle.Price}, {motorcycle.NumWheels}, {motorcycle.EngineSize},{motorcycle.HasSideCar}, {motorcycle.SeatHeight}");
                                break;

                            case "PickupModel":
                                // Cast the vehicle to a pickup
                                PickupModel pickup = (PickupModel)vehicle;
                                // Write the pickup to the file
                                writer.WriteLine($"Pickup, {pickup.Make}, {pickup.Model}, {pickup.Color}, {pickup.Year}, {pickup.Price}, {pickup.NumWheels}, {pickup.EngineSize}, {pickup.HasBedCover}, {pickup.BedSize}");
                                break;

                            default:
                                // Write the vehicle to the file
                                writer.WriteLine($"Vehicle, {vehicle.Make}, {vehicle.Model}, {vehicle.Color}, {vehicle.Year}, {vehicle.Price}, {vehicle.NumWheels}, {vehicle.EngineSize}");
                                break;
                        }
                    }
                }
                // Return if all the data was saved to the file
                return true;
            }
            catch (Exception)
            {
                // Return false if an exception was thrown
                return false;
            }
        } // End of WriteInventory method

        /// <summary>
        /// Read the inventory from a text file
        /// </summary>
        /// <returns></returns>
        public List<VehicleModel> ReadInventory()
        {
            // Declare and Initialize
            string? line = "";
            string[] parts = [];
            string make = "", model = "", color = "";
            int year = 0, numWheels = 0;
            decimal price = 0m, engineSize = 0m;
            // specialty vehicle variables
            bool isConvertible = false, hasSideCar = false, hasBedCover = false;
            decimal trunkSize = 0m, seatHeight = 0m, bedSize = 0m;
```

Figure 26: Challenge Changes



```csharp
        public List<VehicleModel> ReadInventory()
        {
            // Declare and Initialize
            string? line = "";
            string[] parts = [];
            string make = "", model = "", color = "";
            int year = 0, numWheels = 0;
            decimal price = 0m, engineSize = 0m;
            // specialty vehicle variables
            bool isConvertible = false, hasSideCar = false, hasBedCover = false;
            decimal trunkSize = 0m, seatHeight = 0m, bedSize = 0m;

            // Try/Catch any exceptions with the stream reader
            try
            {
                // check if the file exists
                if (File.Exists(_filePath))
                {
                    // Create the stream reader to read from the file
                    using (StreamReader reader = new StreamReader(_filePath))
                    {
                        while ((line = reader.ReadLine()) != null)
                        {
                            // Split the line into parts on a comma-space
                            // Create a string array to put all the seperate vehicle parts into
                            parts = line.Split(", ");

                            // Use the parts array to get the common data (make, model, year,price,numWheels)
                            make = parts[1];
                            model = parts[2];
                            color = parts[3];
                            // Parse the year of the vehicle
                            year = ParseInteger(parts[4]);
                            // Parse the price of the vehicle
                            price = ParseDecimal(parts[5]);
                            // Parse the number of wheels
                            numWheels = ParseInteger(parts[6]);
                            // Parse the engine size
                            engineSize = ParseDecimal(parts[7]);

                            // Use the first part of the data to create a switch for the specific model
                            switch (parts[0])
                            {
                                case "Car":
                                    // Parse the convertible status for the car
                                    isConvertible = ParseBoolean(parts[8]);
                                    // Parse the trunk size for the car
                                    trunkSize = ParseDecimal(parts[9]);
                                    // Create a new car model and add it to the inventory
                                    CarModel car = new CarModel(0, make, model, color, year, price, numWheels, engineSize, isConvertible, trunkSize);
                                    // Add the car to the inventory
                                    AddVehicleToInventory(car);
                                    break;

                                case "Motorcycle":
                                    // Parse the side car status for the motorcycle
                                    hasSideCar = ParseBoolean(parts[8]);
                                    // Parse the seat height for the motorcycle
                                    seatHeight = ParseDecimal(parts[9]);
                                    // Create a new motorcycle using the read properties
                                    MotorcycleModel motorcycle = new MotorcycleModel(0, make, model, color, year, price, numWheels, engineSize, hasSideCar, seatHeight);
                                    // Add the motorcycle to the inventory
                                    AddVehicleToInventory(motorcycle);
                                    break;

                                case "Pickup":
                                    // Parse the bed cover status for the pickup
                                    hasBedCover = ParseBoolean(parts[8]);
                                    // Parse the bed size for the pickup
                                    bedSize = ParseDecimal(parts[9]);
                                    // Create a new pickup using the read properties
                                    PickupModel pickup = new PickupModel(0, make, model, color, year, price, numWheels, engineSize, hasBedCover, bedSize);
                                    AddVehicleToInventory(pickup);
                                    break;

                                default:
                                    // Create a new vehicle using the read properties
                                    VehicleModel vehicle = new VehicleModel(0, make, model, color, year, price, numWheels, engineSize);
                                    AddVehicleToInventory(vehicle);
                                    break;
                            }
```

Figure 27 Challenge changes



```
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
3
Enter 1 to create a car, 2 to create a motorcycle, 3 to create a pickup, or 4 to create a vehicle: 1
Enter the make of the vehicle: Ford
Enter the model of the vehicle: Bronco
Enter the color of the vehicle: Blue
Enter the year of the vehicle: 2020
Enter the price of the vehicle: 62000
Enter the number of wheels on the vehicle: 4
Enter the engine size of the vehicle in liters: 3.6
Enter if the car is a convertible (true/false): true
Enter the trunk size of the car in cubic feet: 2.2

Chose an action:
0) Quit
1) Print the Inventory
2) Print the Shopping Cart
3) Create a Vehicle
4) Add a Vehicle to the Shopping Cart
5) Checkout.
6) Save inventory to a text file.
7) Load inventory from a text file.
Input:
```

Figure 28: Changes running

Figures 26-28 are the changes to add the new properties and the application running with the new properties added to the create vehicle screens

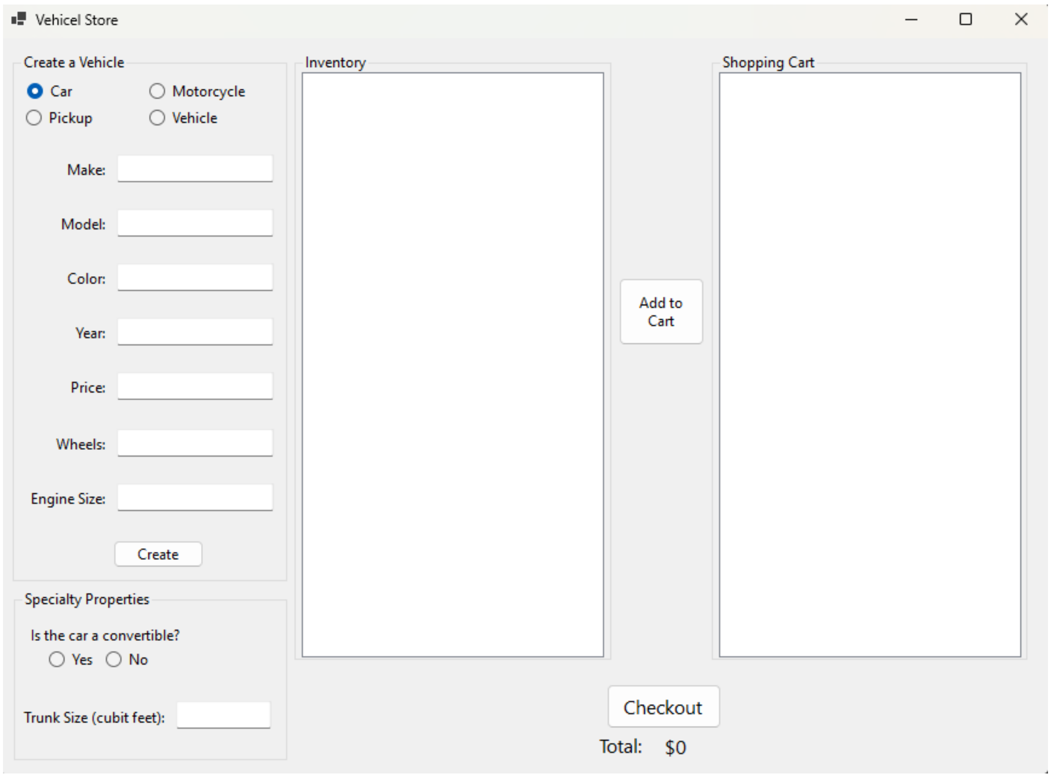
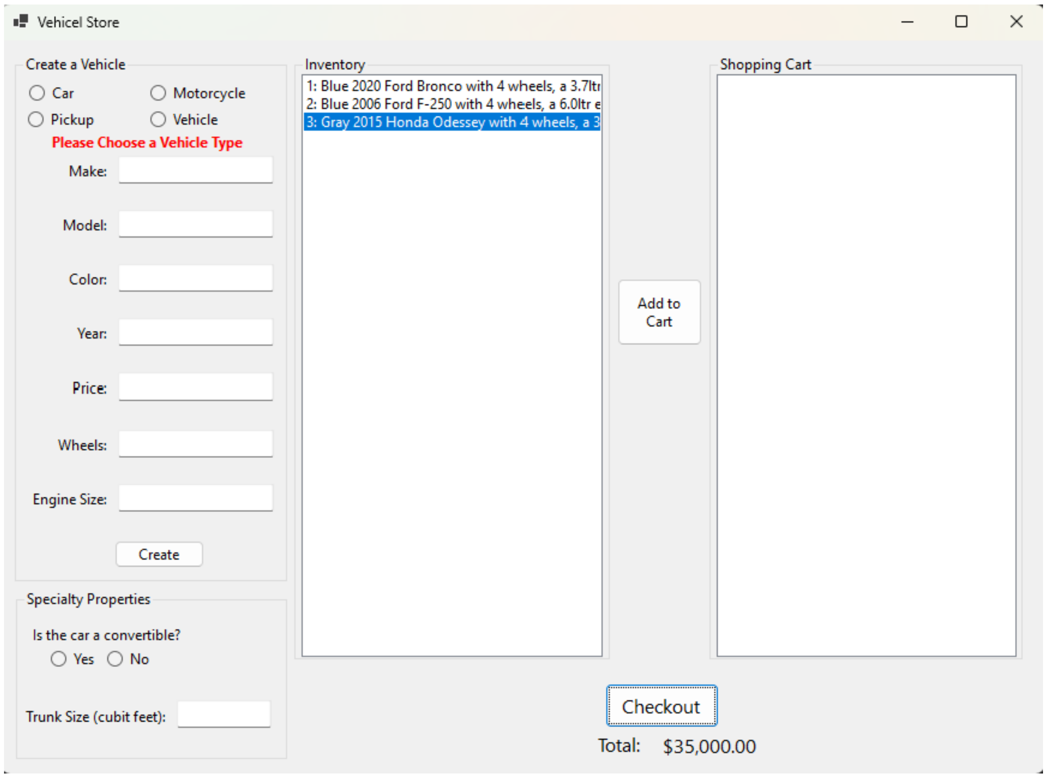


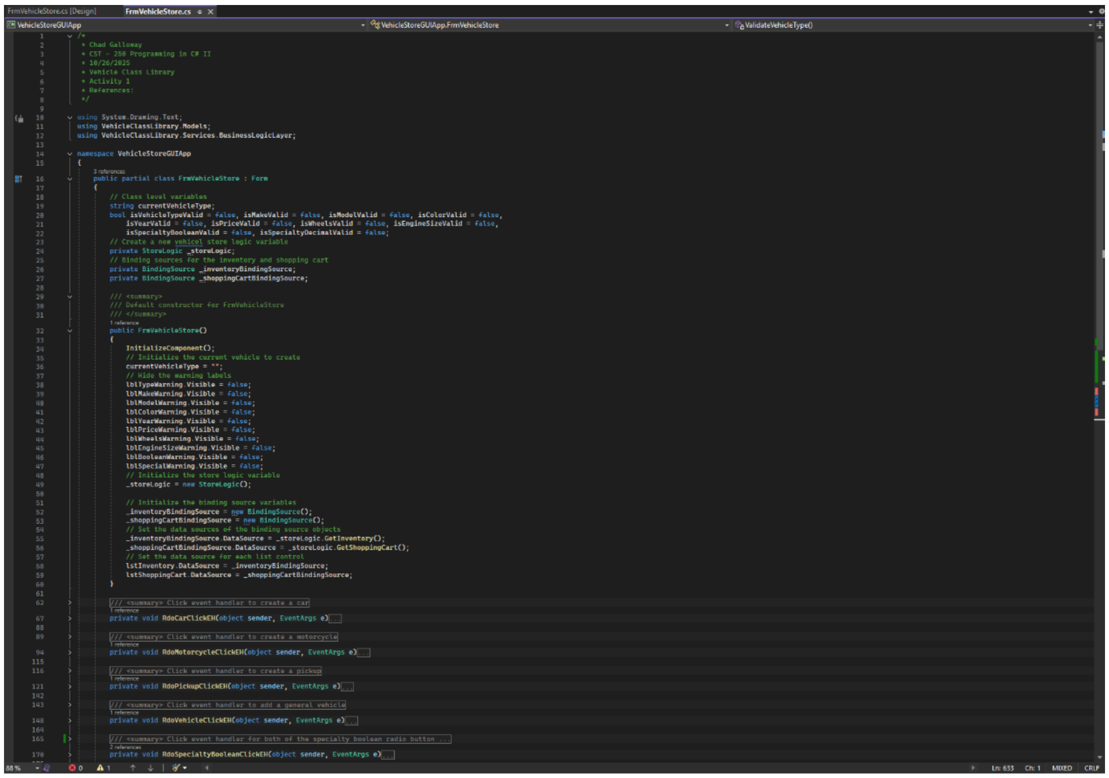
Figure 29: Initial state of UI



```
1: Blue 2020 Ford Bronco with 4 wheels, a 3.7ltr
2: Blue 2006 Ford F-250 with 4 wheels, a 6.0ltr e
3: Gray 2015 Honda Odessey with 4 wheels, a 3
```

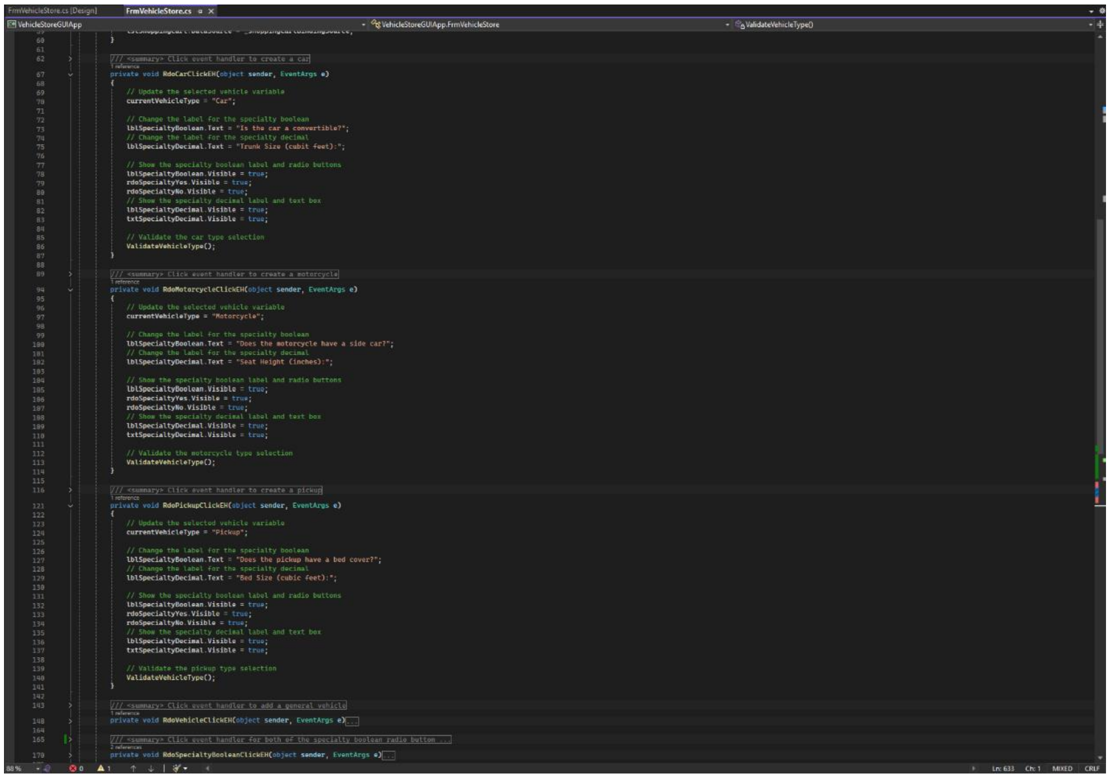
Total: $35,000.00

Figure 30: Ui after creating some vehicles and checking one out


FrmVehicleStore.cs [Design]
FrmVehicleStore.cs
VehicleStoreGUIApp
VehicleStoreGUIApp.FrmVehicleStore
ValidateVehicleType()

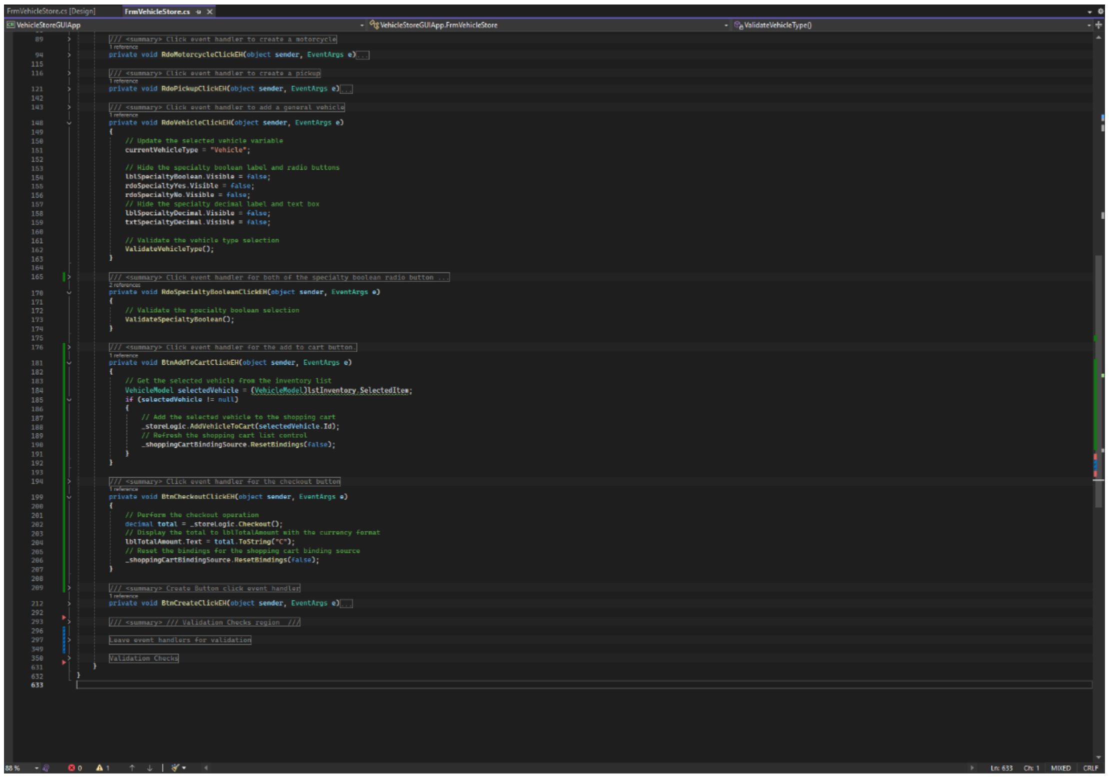
```
/*
 * Chad Galloway
 * CST - 250 Programming in C# II
 * 10/26/2025
 * Vehicle Class Library
 * Activity 1
 * References:
 */

using System.Drawing.Text;
using VehicleClassLibrary.Models;
using VehicleClassLibrary.Services.BusinessLogicLayer;

namespace VehicleStoreGUIApp
{
    public partial class FrmVehicleStore : Form
    {
        // Class level variables
        string currentVehicleType;
        bool isVehicleTypeValid = false, isMakeValid = false, isModelValid = false, isColorValid = false,
            isYearValid = false, isPriceValid = false, isWheelsValid = false, isEngineSizeValid = false,
            isSpecialtyBooleanValid = false, isSpecialtyDecimalValid = false;
        // Create a new vehicel store logic variable
        private StoreLogic _storeLogic;
        // Binding sources for the inventory and shopping cart
        private BindingSource _inventoryBindingSource;
        private BindingSource _shoppingCartBindingSource;

        /// <summary>
        /// Default constructor for FrmVehicleStore
        /// </summary>
        public FrmVehicleStore()
        {
            InitializeComponent();
            // Initialize the current vehicle to create
            currentVehicleType = "";
            // Hide the warning labels
            lblTypeWarning.Visible = false;
            lblMakeWarning.Visible = false;
            lblModelWarning.Visible = false;
            lblColorWarning.Visible = false;
            lblYearWarning.Visible = false;
            lblPriceWarning.Visible = false;
            lblWheelsWarning.Visible = false;
            lblEngineSizeWarning.Visible = false;
            lblBooleanWarning.Visible = false;
            lblSpecialWarning.Visible = false;
            // Initialize the store logic variable
            _storeLogic = new StoreLogic();

            // Initialize the binding source variables
            _inventoryBindingSource = new BindingSource();
            _shoppingCartBindingSource = new BindingSource();
            // Set the data sources of the binding source objects
            _inventoryBindingSource.DataSource = _storeLogic.GetInventory();
            _shoppingCartBindingSource.DataSource = _storeLogic.GetShoppingCart();
            // Set the data source for each list control
            lstInventory.DataSource = _inventoryBindingSource;
            lstShoppingCart.DataSource = _shoppingCartBindingSource;
        }

        /// <summary> Click event handler to create a car
        private void RdoCarClickEH(object sender, EventArgs e)...

        /// <summary> Click event handler to create a motorcycle
        private void RdoMotorcycleClickEH(object sender, EventArgs e)...

        /// <summary> Click event handler to create a pickup
        private void RdoPickupClickEH(object sender, EventArgs e)...

        /// <summary> Click event handler to add a general vehicle
        private void RdoVehicleClickEH(object sender, EventArgs e)...

        /// <summary> Click event handler for both of the specialty boolean radio button ...
        private void RdoSpecialtyBooleanClickEH(object sender, EventArgs e)...
```

Figure 31: The FrmVehicleStore class declaration, citations, and constructors



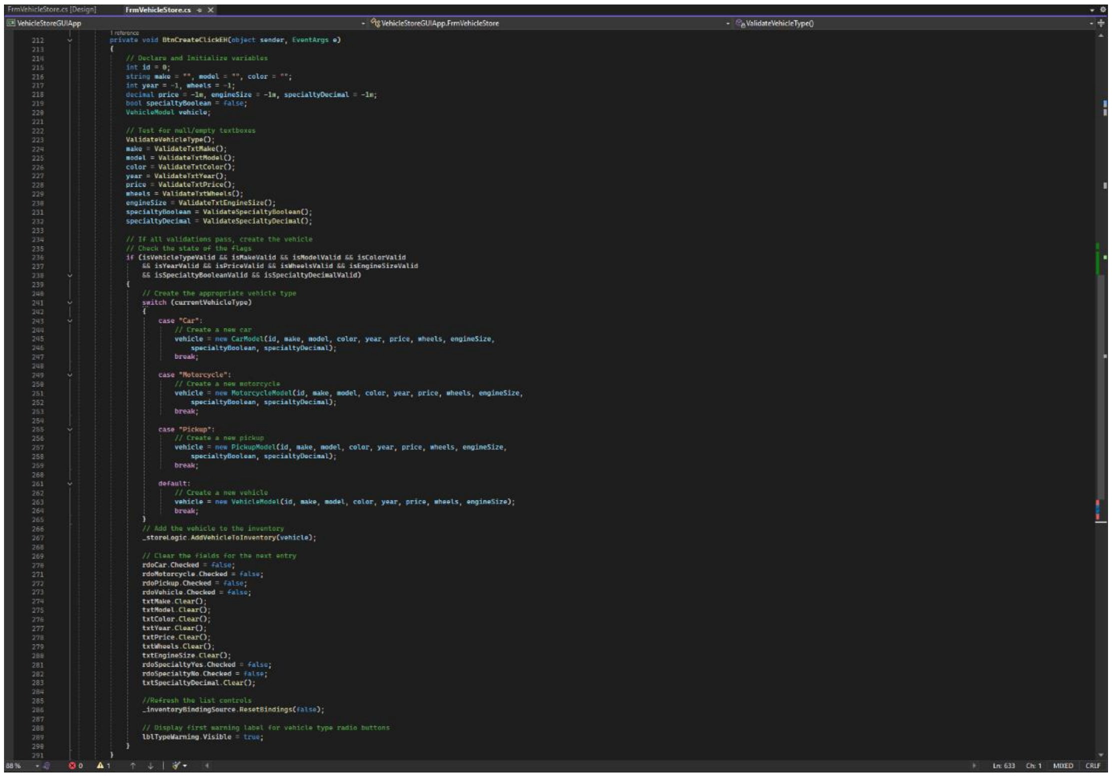
```
FrmVehicleStore.cs [Design]    FrmVehicleStore.cs
VehicleStoreGUIApp    VehicleStoreGUIApp.FrmVehicleStore    ValidateVehicleType()

        }

        /// <summary> Click event handler to create a car
        1 reference
        private void RdoCarClickEH(object sender, EventArgs e)
        {
            // Update the selected vehicle variable
            currentVehicleType = "Car";

            // Change the label for the specialty boolean
            lblSpecialtyBoolean.Text = "Is the car a convertible?";
            // Change the label for the specialty decimal
            lblSpecialtyDecimal.Text = "Trunk Size (cubit feet):";

            // Show the specialty boolean label and radio buttons
            lblSpecialtyBoolean.Visible = true;
            rdoSpecialtyYes.Visible = true;
            rdoSpecialtyNo.Visible = true;
            // Show the specialty decimal label and text box
            lblSpecialtyDecimal.Visible = true;
            txtSpecialtyDecimal.Visible = true;

            // Validate the car type selection
            ValidateVehicleType();
        }

        /// <summary> Click event handler to create a motorcycle
        1 reference
        private void RdoMotorcycleClickEH(object sender, EventArgs e)
        {
            // Update the selected vehicle variable
            currentVehicleType = "Motorcycle";

            // Change the label for the specialty boolean
            lblSpecialtyBoolean.Text = "Does the motorcycle have a side car?";
            // Change the label for the specialty decimal
            lblSpecialtyDecimal.Text = "Seat Height (inches):";

            // Show the specialty boolean label and radio buttons
            lblSpecialtyBoolean.Visible = true;
            rdoSpecialtyYes.Visible = true;
            rdoSpecialtyNo.Visible = true;
            // Show the specialty decimal label and text box
            lblSpecialtyDecimal.Visible = true;
            txtSpecialtyDecimal.Visible = true;

            // Validate the motorcycle type selection
            ValidateVehicleType();
        }

        /// <summary> Click event handler to create a pickup
        1 reference
        private void RdoPickupClickEH(object sender, EventArgs e)
        {
            // Update the selected vehicle variable
            currentVehicleType = "Pickup";

            // Change the label for the specialty boolean
            lblSpecialtyBoolean.Text = "Does the pickup have a bed cover?";
            // Change the label for the specialty decimal
            lblSpecialtyDecimal.Text = "Bed Size (cubic feet):";

            // Show the specialty boolean label and radio buttons
            lblSpecialtyBoolean.Visible = true;
            rdoSpecialtyYes.Visible = true;
            rdoSpecialtyNo.Visible = true;
            // Show the specialty decimal label and text box
            lblSpecialtyDecimal.Visible = true;
            txtSpecialtyDecimal.Visible = true;

            // Validate the pickup type selection
            ValidateVehicleType();
        }

        /// <summary> Click event handler to add a general vehicle
        1 reference
        private void RdoVehicleClickEH(object sender, EventArgs e)...

        /// <summary> Click event handler for both of the specialty boolean radio button ...
        2 references
        private void RdoSpecialtyBooleanClickEH(object sender, EventArgs e)...
88 %    0    1                                        Ln: 633   Ch: 1   MIXED   CRLF
```

Figure 31: event handlers for Car, Motorcycle, and Pickup radio buttons



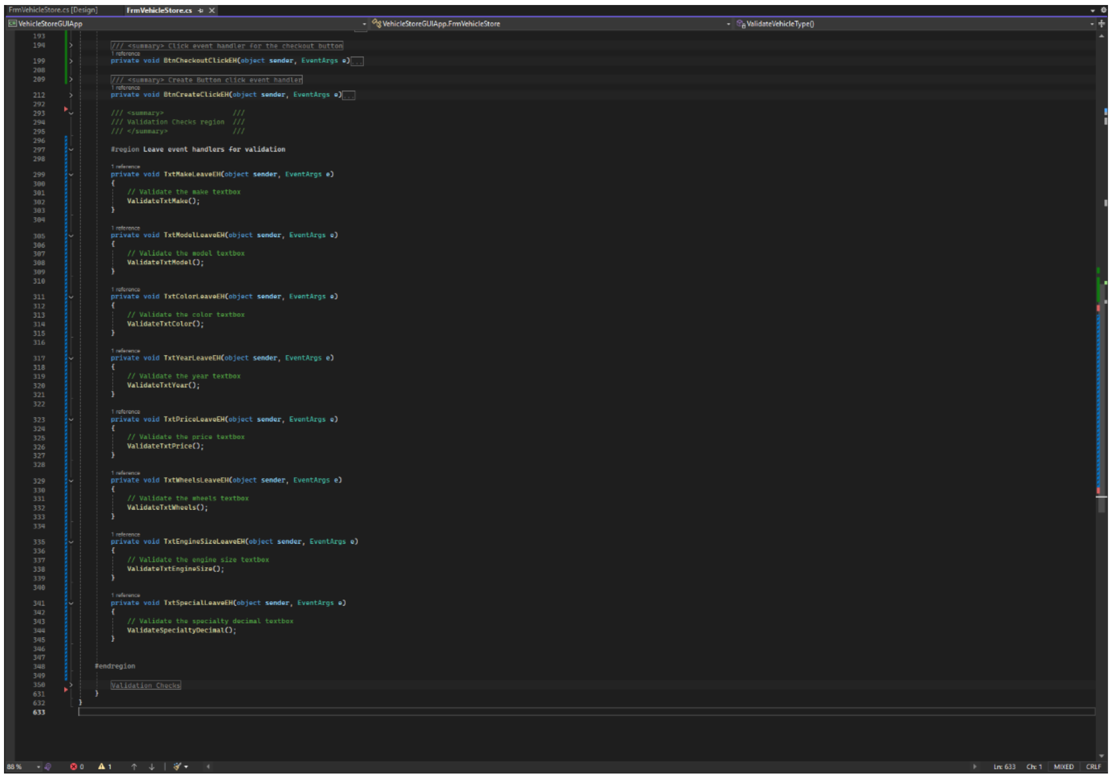
```
FrmVehicleStore.cs [Design]    FrmVehicleStore.cs
VehicleStoreGUIApp    VehicleStoreGUIApp.FrmVehicleStore    ValidateVehicleType()
        /// <summary> Click event handler to create a motorcycle
        1 reference
        private void RdoMotorcycleClickEH(object sender, EventArgs e)...

        /// <summary> Click event handler to create a pickup
        1 reference
        private void RdoPickupClickEH(object sender, EventArgs e)...

        /// <summary> Click event handler to add a general vehicle
        1 reference
        private void RdoVehicleClickEH(object sender, EventArgs e)
        {
            // Update the selected vehicle variable
            currentVehicleType = "Vehicle";

            // Hide the specialty boolean label and radio buttons
            lblSpecialtyBoolean.Visible = false;
            rdoSpecialtyYes.Visible = false;
            rdoSpecialtyNo.Visible = false;
            // Hide the specialty decimal label and text box
            lblSpecialtyDecimal.Visible = false;
            txtSpecialtyDecimal.Visible = false;

            // Validate the vehicle type selection
            ValidateVehicleType();
        }

        /// <summary> Click event handler for both of the specialty boolean radio button ...
        2 references
        private void RdoSpecialtyBooleanClickEH(object sender, EventArgs e)
        {
            // Validate the specialty boolean selection
            ValidateSpecialtyBoolean();
        }

        /// <summary> Click event handler for the add to cart button.
        1 reference
        private void BtnAddToCartClickEH(object sender, EventArgs e)
        {
            // Get the selected vehicle from the inventory list
            VehicleModel selectedVehicle = (VehicleModel)lstInventory.SelectedItem;
            if (selectedVehicle != null)
            {
                // Add the selected vehicle to the shopping cart
                _storeLogic.AddVehicleToCart(selectedVehicle.Id);
                // Refresh the shopping cart list control
                _shoppingCartBindingSource.ResetBindings(false);
            }
        }

        /// <summary> Click event handler for the checkout button
        1 reference
        private void BtnCheckoutClickEH(object sender, EventArgs e)
        {
            // Perform the checkout operation
            decimal total = _storeLogic.Checkout();
            // Display the total to lblTotalAmount with the currency format
            lblTotalAmount.Text = total.ToString("C");
            // Reset the bindings for the shopping cart binding source
            _shoppingCartBindingSource.ResetBindings(false);
        }

        /// <summary> Create Button click event handler
        1 reference
        private void BtnCreateClickEH(object sender, EventArgs e)...

        /// <summary> /// Validation Checks region  ///

        Leave event handlers for validation

        Validation Checks
    }
}

88 %    0    1                                        Ln: 633   Ch: 1   MIXED   CRLF
```

Figure 32: Event handlers for the remaining two radio buttons and the add to cart and checkout buttons.



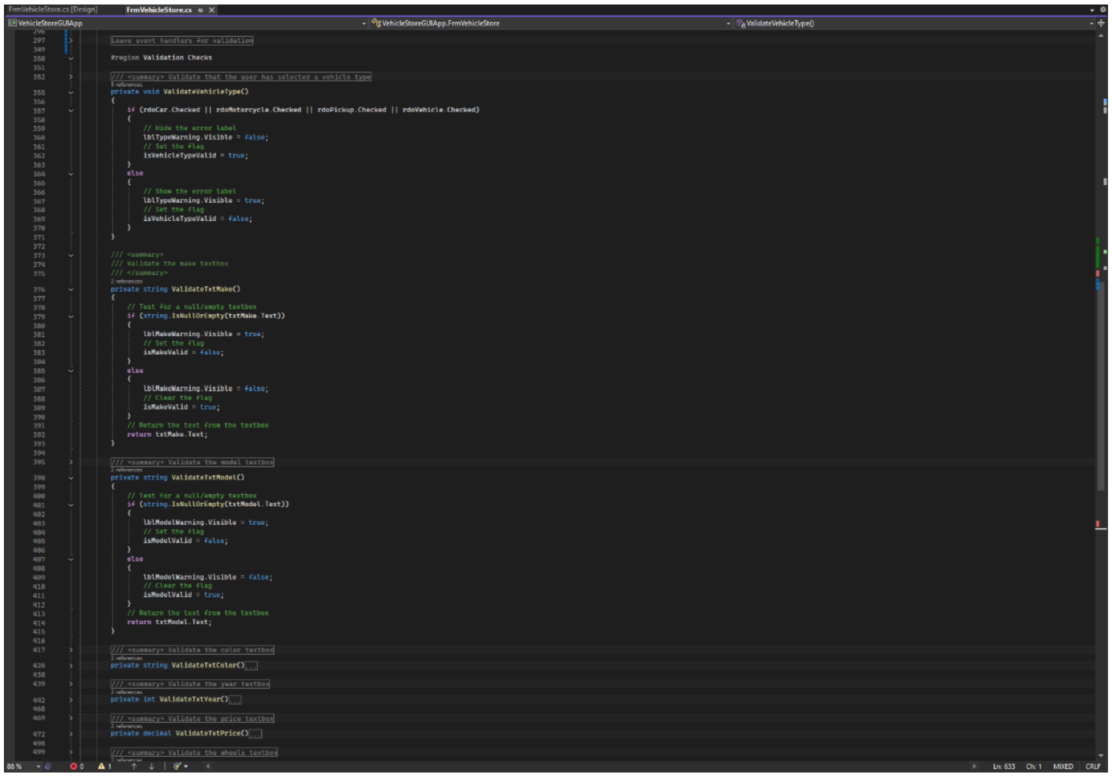
```
private void BtnCreateClickEH(object sender, EventArgs e)
{
    // Declare and Initialize variables
    int id = 0;
    string make = "", model = "", color = "";
    int year = -1, wheels = -1;
    decimal price = -1m, engineSize = -1m, specialtyDecimal = -1m;
    bool specialtyBoolean = false;
    VehicleModel vehicle;

    // Test for null/empty textboxes
    ValidateVehicleType();
    make = ValidateTxtMake();
    model = ValidateTxtModel();
    color = ValidateTxtColor();
    year = ValidateTxtYear();
    price = ValidateTxtPrice();
    wheels = ValidateTxtWheels();
    engineSize = ValidateTxtEngineSize();
    specialtyBoolean = ValidateSpecialtyBoolean();
    specialtyDecimal = ValidateSpecialtyDecimal();

    // If all validations pass, create the vehicle
    // Check the state of the flags
    if (isVehicleTypeValid && isMakeValid && isModelValid && isColorValid
        && isYearValid && isPriceValid && isWheelsValid && isEngineSizeValid
        && isSpecialtyBooleanValid && isSpecialtyDecimalValid)
    {
        // Create the appropriate vehicle type
        switch (currentVehicleType)
        {
            case "Car":
                // Create a new car
                vehicle = new CarModel(id, make, model, color, year, price, wheels, engineSize,
                    specialtyBoolean, specialtyDecimal);
                break;

            case "Motorcycle":
                // Create a new motorcycle
                vehicle = new MotorcycleModel(id, make, model, color, year, price, wheels, engineSize,
                    specialtyBoolean, specialtyDecimal);
                break;

            case "Pickup":
                // Create a new pickup
                vehicle = new PickupModel(id, make, model, color, year, price, wheels, engineSize,
                    specialtyBoolean, specialtyDecimal);
                break;

            default:
                // Create a new vehicle
                vehicle = new VehicleModel(id, make, model, color, year, price, wheels, engineSize);
                break;
        }
        // Add the vehicle to the inventory
        _storeLogic.AddVehicleToInventory(vehicle);

        // Clear the fields for the next entry
        rdoCar.Checked = false;
        rdoMotorcycle.Checked = false;
        rdoPickup.Checked = false;
        rdoVehicle.Checked = false;
        txtMake.Clear();
        txtModel.Clear();
        txtColor.Clear();
        txtYear.Clear();
        txtPrice.Clear();
        txtWheels.Clear();
        txtEngineSize.Clear();
        rdoSpecialtyYes.Checked = false;
        rdoSpecialtyNo.Checked = false;
        txtSpecialtyDecimal.Clear();

        //Refresh the list controls
        _inventoryBindingSource.ResetBindings(false);

        // Display first warning label for vehicle type radio buttons
        lblTypeWarning.Visible = true;
    }
}
```

Figure 33: Create Button click event handler.



```
/// <summary> Click event handler for the checkout button
private void BtnCheckoutClickEH(object sender, EventArgs e)...

/// <summary> Create Button click event handler
private void BtnCreateClickEH(object sender, EventArgs e)...

/// <summary>                ///
/// Validation Checks region  ///
/// </summary>               ///

#region Leave event handlers for validation

private void TxtMakeLeaveEH(object sender, EventArgs e)
{
    // Validate the make textbox
    ValidateTxtMake();
}

private void TxtModelLeaveEH(object sender, EventArgs e)
{
    // Validate the model textbox
    ValidateTxtModel();
}

private void TxtColorLeaveEH(object sender, EventArgs e)
{
    // Validate the color textbox
    ValidateTxtColor();
}

private void TxtYearLeaveEH(object sender, EventArgs e)
{
    // Validate the year textbox
    ValidateTxtYear();
}

private void TxtPriceLeaveEH(object sender, EventArgs e)
{
    // Validate the price textbox
    ValidateTxtPrice();
}

private void TxtWheelsLeaveEH(object sender, EventArgs e)
{
    // Validate the wheels textbox
    ValidateTxtWheels();
}

private void TxtEngineSizeLeaveEH(object sender, EventArgs e)
{
    // Validate the engine size textbox
    ValidateTxtEngineSize();
}

private void TxtSpecialLeaveEH(object sender, EventArgs e)
{
    // Validate the specialty decimal textbox
    ValidateSpecialtyDecimal();
}


#endregion

    Validation Checks
}
}
```

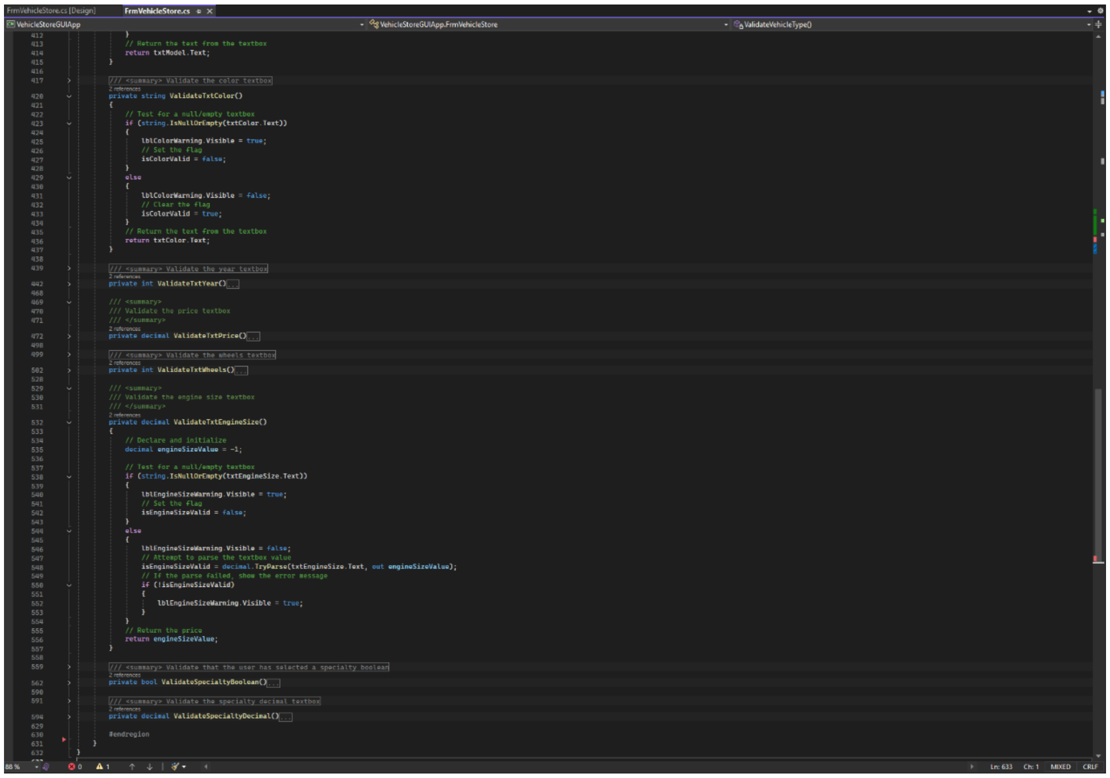
Figure 34: All the leave event handlers



```
Leave event handlers for validation

#region Validation Checks

/// <summary> Validate that the user has selected a vehicle type
private void ValidateVehicleType()
{
    if (rdoCar.Checked || rdoMotorcycle.Checked || rdoPickup.Checked || rdoVehicle.Checked)
    {
        // Hide the error label
        lblTypeWarning.Visible = false;
        // Set the flag
        isVehicleTypeValid = true;
    }
    else
    {
        // Show the error label
        lblTypeWarning.Visible = true;
        // Set the flag
        isVehicleTypeValid = false;
    }
}

/// <summary>
/// Validate the make textbox
/// </summary>
private string ValidateTxtMake()
{
    // Test for a null/empty textbox
    if (string.IsNullOrEmpty(txtMake.Text))
    {
        lblMakeWarning.Visible = true;
        // Set the flag
        isMakeValid = false;
    }
    else
    {
        lblMakeWarning.Visible = false;
        // Clear the flag
        isMakeValid = true;
    }
    // Return the text from the textbox
    return txtMake.Text;
}

/// <summary> Validate the model textbox
private string ValidateTxtModel()
{
    // Test for a null/empty textbox
    if (string.IsNullOrEmpty(txtModel.Text))
    {
        lblModelWarning.Visible = true;
        // Set the flag
        isModelValid = false;
    }
    else
    {
        lblModelWarning.Visible = false;
        // Clear the flag
        isModelValid = true;
    }
    // Return the text from the textbox
    return txtModel.Text;
}

/// <summary> Validate the color textbox
private string ValidateTxtColor()...

/// <summary> Validate the year textbox
private int ValidateTxtYear()...

/// <summary> Validate the price textbox
private decimal ValidateTxtPrice()...

/// <summary> Validate the wheels textbox
```

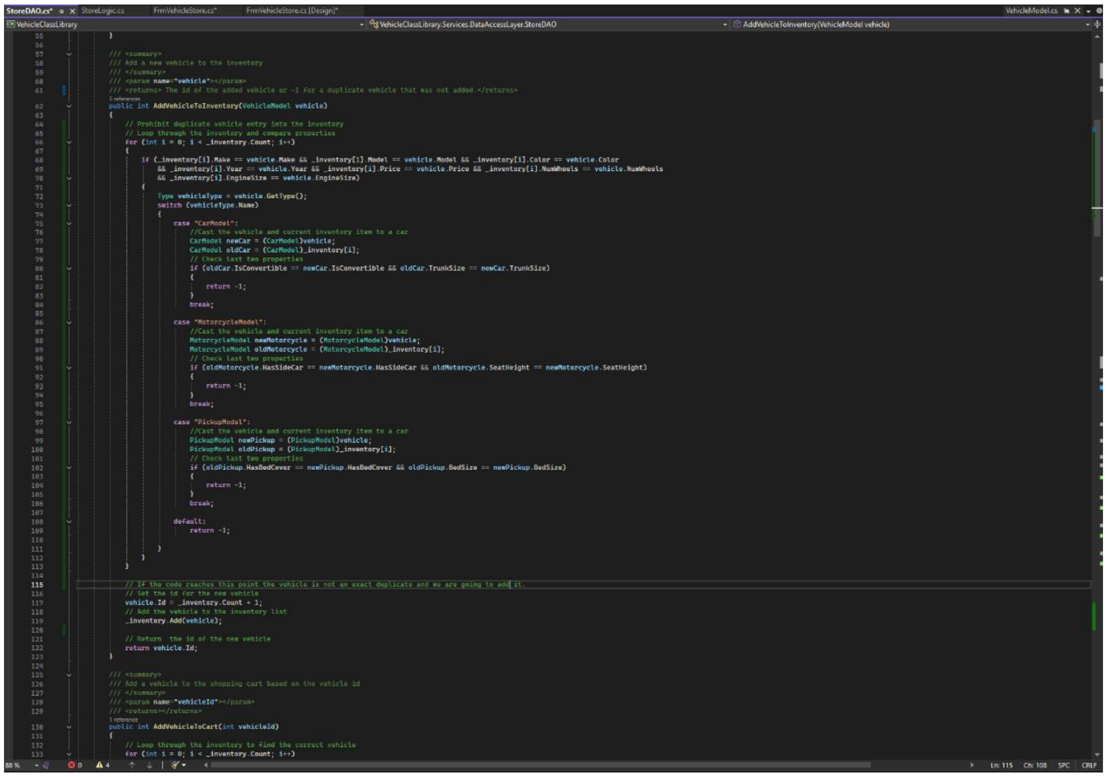
Figure 35: Textbox and radio button validation checks

Part 2



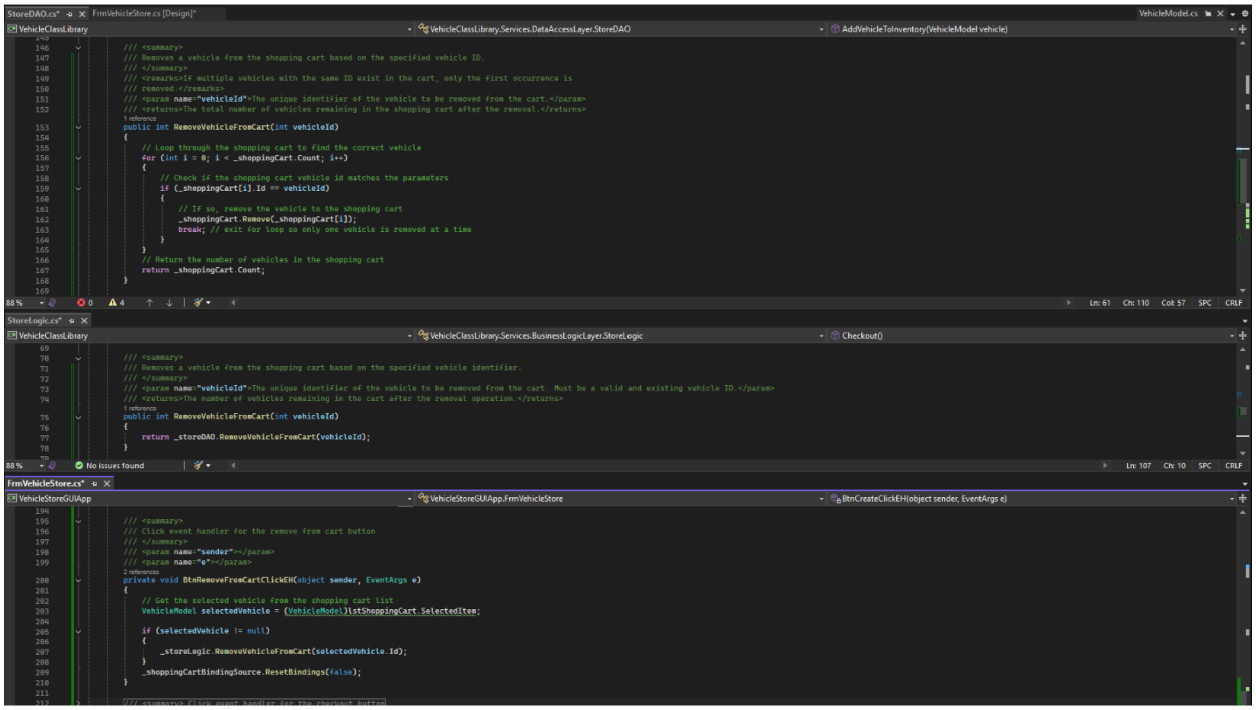
```
            }
            // Return the text from the textbox
            return txtModel.Text;
        }

        /// <summary> Validate the color textbox
        2 references
        private string ValidateTxtColor()
        {
            // Test for a null/empty textbox
            if (string.IsNullOrEmpty(txtColor.Text))
            {
                lblColorWarning.Visible = true;
                // Set the flag
                isColorValid = false;
            }
            else
            {
                lblColorWarning.Visible = false;
                // Clear the flag
                isColorValid = true;
            }
            // Return the text from the textbox
            return txtColor.Text;
        }

        /// <summary> Validate the year textbox
        2 references
        private int ValidateTxtYear()...

        /// <summary>
        /// Validate the price textbox
        /// </summary>
        2 references
        private decimal ValidateTxtPrice()...

        /// <summary> Validate the wheels textbox
        2 references
        private int ValidateTxtWheels()...

        /// <summary>
        /// Validate the engine size textbox
        /// </summary>
        2 references
        private decimal ValidateTxtEngineSize()
        {
            // Declare and initialize
            decimal engineSizeValue = -1;

            // Test for a null/empty textbox
            if (string.IsNullOrEmpty(txtEngineSize.Text))
            {
                lblEngineSizeWarning.Visible = true;
                // Set the flag
                isEngineSizeValid = false;
            }
            else
            {
                lblEngineSizeWarning.Visible = false;
                // Attempt to parse the textbox value
                isEngineSizeValid = decimal.TryParse(txtEngineSize.Text, out engineSizeValue);
                // If the parse failed, show the error message
                if (!isEngineSizeValid)
                {
                    lblEngineSizeWarning.Visible = true;
                }
            }
            // Return the price
            return engineSizeValue;
        }

        /// <summary> Validate that the user has selected a specialty boolean
        2 references
        private bool ValidateSpecialtyBoolean()...

        /// <summary> Validate the specialty decimal textbox
        2 references
        private decimal ValidateSpecialtyDecimal()...

        #endregion
    }
}
```

Figure 36: New property validation checks



```
        }

        /// <summary>
        /// Add a new vehicle to the inventory
        /// </summary>
        /// <param name="vehicle"></param>
        /// <returns> The id of the added vehicle or -1 for a duplicate vehicle that was not added.</returns>
        5 references
        public int AddVehicleToInventory(VehicleModel vehicle)
        {
            // Prohibit duplicate vehicle entry into the inventory
            // Loop through the inventory and compare properties
            for (int i = 0; i < _inventory.Count; i++)
            {
                if (_inventory[i].Make == vehicle.Make && _inventory[i].Model == vehicle.Model && _inventory[i].Color == vehicle.Color
                    && _inventory[i].Year == vehicle.Year && _inventory[i].Price == vehicle.Price && _inventory[i].NumWheels == vehicle.NumWheels
                    && _inventory[i].EngineSize == vehicle.EngineSize)
                {
                    Type vehicleType = vehicle.GetType();
                    switch (vehicleType.Name)
                    {
                        case "CarModel":
                            //Cast the vehicle and current inventory item to a car
                            CarModel newCar = (CarModel)vehicle;
                            CarModel oldCar = (CarModel)_inventory[i];
                            // Check last two properties
                            if (oldCar.IsConvertible == newCar.IsConvertible && oldCar.TrunkSize == newCar.TrunkSize)
                            {
                                return -1;
                            }
                            break;

                        case "MotorcycleModel":
                            //Cast the vehicle and current inventory item to a car
                            MotorcycleModel newMotorcycle = (MotorcycleModel)vehicle;
                            MotorcycleModel oldMotorcycle = (MotorcycleModel)_inventory[i];
                            // Check last two properties
                            if (oldMotorcycle.HasSideCar == newMotorcycle.HasSideCar && oldMotorcycle.SeatHeight == newMotorcycle.SeatHeight)
                            {
                                return -1;
                            }
                            break;

                        case "PickupModel":
                            //Cast the vehicle and current inventory item to a car
                            PickupModel newPickup = (PickupModel)vehicle;
                            PickupModel oldPickup = (PickupModel)_inventory[i];
                            // Check last two properties
                            if (oldPickup.HasBedCover == newPickup.HasBedCover && oldPickup.BedSize == newPickup.BedSize)
                            {
                                return -1;
                            }
                            break;

                        default:
                            return -1;

                    }
                }
            }

            // If the code reaches this point the vehicle is not an exact duplicate and we are going to add it.
            // Set the id for the new vehicle
            vehicle.Id = _inventory.Count + 1;
            // Add the vehicle to the inventory list
            _inventory.Add(vehicle);

            // Return  the id of the new vehicle
            return vehicle.Id;
        }

        /// <summary>
        /// Add a vehicle to the shopping cart based on the vehicle id
        /// </summary>
        /// <param name="vehicleId"></param>
        /// <returns></returns>
        1 reference
        public int AddVehicleToCart(int vehicleId)
        {
            // Loop through the inventory to find the correct vehicle
            for (int i = 0; i < _inventory.Count; i++)
```

Figure 37: new code added to StoreDAO to prevent duplicates

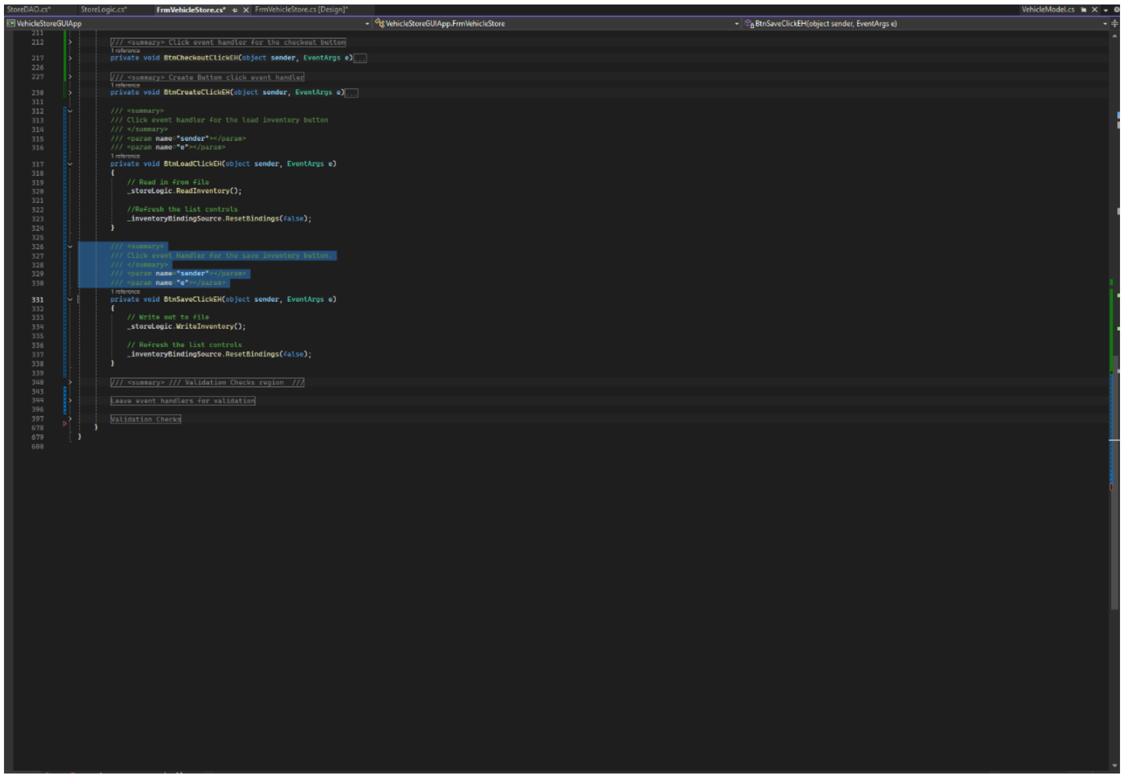


```
/// <summary>
/// Removes a vehicle from the shopping cart based on the specified vehicle ID.
/// </summary>
/// <remarks>If multiple vehicles with the same ID exist in the cart, only the first occurrence is
/// removed.</remarks>
/// <param name="vehicleId">The unique identifier of the vehicle to be removed from the cart.</param>
/// <returns>The total number of vehicles remaining in the shopping cart after the removal.</returns>
public int RemoveVehicleFromCart(int vehicleId)
{
    // Loop through the shopping cart to find the correct vehicle
    for (int i = 0; i < _shoppingCart.Count; i++)
    {
        // Check if the shopping cart vehicle id matches the parameters
        if (_shoppingCart[i].Id == vehicleId)
        {
            // If so, remove the vehicle to the shopping cart
            _shoppingCart.Remove(_shoppingCart[i]);
            break; // exit for loop so only one vehicle is removed at a time
        }
    }
    // Return the number of vehicles in the shopping cart
    return _shoppingCart.Count;
}
```

```
/// <summary>
/// Removes a vehicle from the shopping cart based on the specified vehicle identifier.
/// </summary>
/// <param name="vehicleId">The unique identifier of the vehicle to be removed from the cart. Must be a valid and existing vehicle ID.</param>
/// <returns>The number of vehicles remaining in the cart after the removal operation.</returns>
public int RemoveVehicleFromCart(int vehicleId)
{
    return _storeDAO.RemoveVehicleFromCart(vehicleId);
}
```

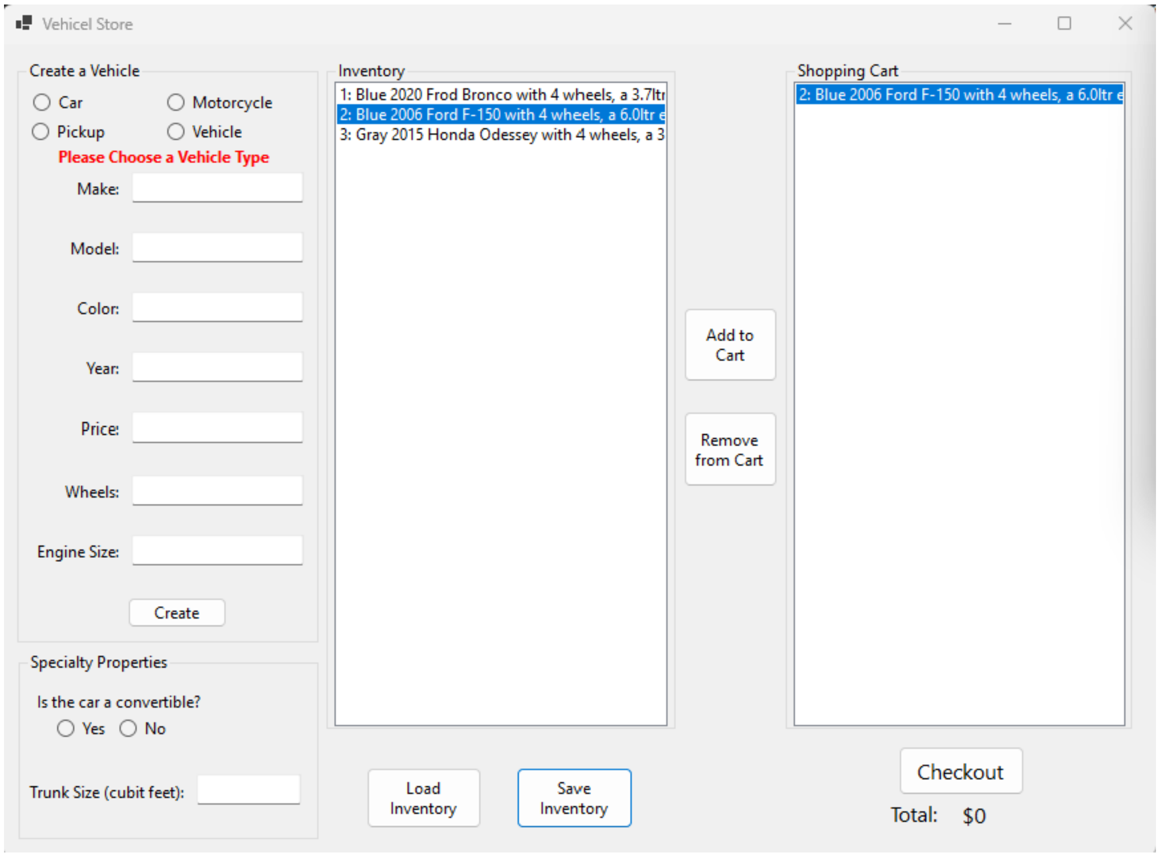
```
/// <summary>
/// Click event handler for the remove from cart button
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void BtnRemoveFromCartClickEH(object sender, EventArgs e)
{
    // Get the selected vehicle from the shopping cart list
    VehicleModel selectedVehicle = (VehicleModel)lstShoppingCart.SelectedItem;

    if (selectedVehicle != null)
    {
        _storeLogic.RemoveVehicleFromCart(selectedVehicle.Id);
    }
    _shoppingCartBindingSource.ResetBindings(false);
}
/// <summary> Click event handler for the checkout button
```

Figure 38: The three methods responsible for removing a vehicle from the cart in all three layers, the StoreDAO, the StoreLogic and the FrmVehicleStore



```
/// <summary> Click event handler for the checkout button
private void BtnCheckoutClickEH(object sender, EventArgs e)...

/// <summary> Create Button click event handler
private void BtnCreateClickEH(object sender, EventArgs e)...

/// <summary>
/// Click event handler for the load inventory button
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void BtnLoadClickEH(object sender, EventArgs e)
{
    // Read in from file
    _storeLogic.ReadInventory();

    //Refresh the list controls
    _inventoryBindingSource.ResetBindings(false);
}

/// <summary>
/// Click event Handler for the save inventory button.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void BtnSaveClickEH(object sender, EventArgs e)
{
    // Write out to file
    _storeLogic.WriteInventory();

    // Refresh the list controls
    _inventoryBindingSource.ResetBindings(false);
}

/// <summary> /// Validation Checks region  ///

Leave event handlers for validation

Validation Checks
    }
}
```

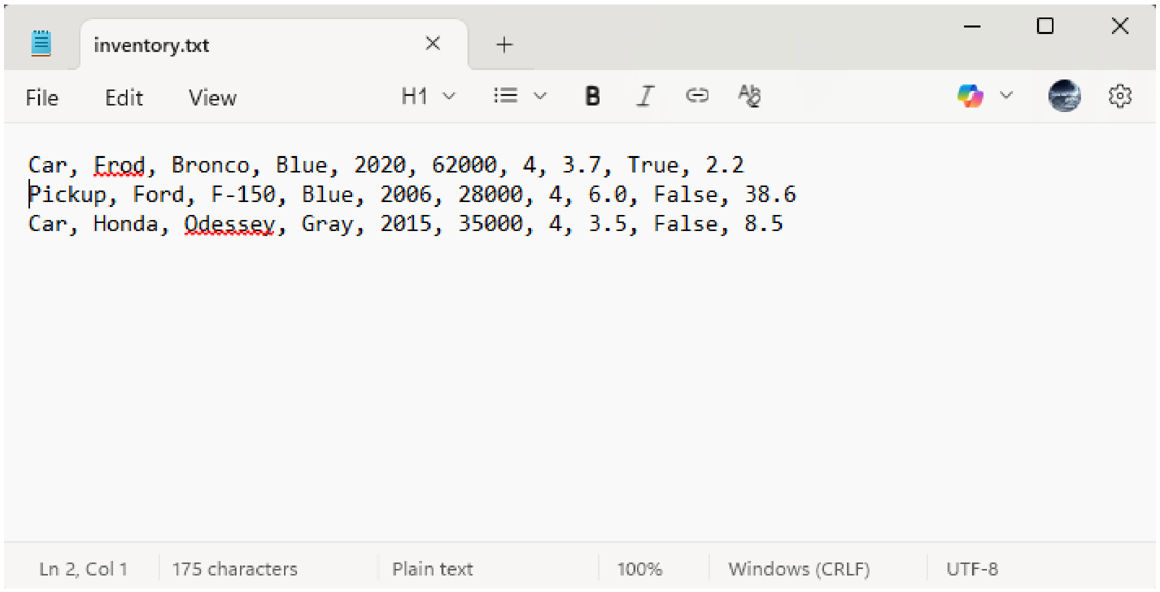
Figure 39: Implementation of the save and load functionality for the GUI



```
1: Blue 2020 Frod Bronco with 4 wheels, a 3.7ltr
2: Blue 2006 Ford F-150 with 4 wheels, a 6.0ltr e
3: Gray 2015 Honda Odessey with 4 wheels, a 3
```

```
2: Blue 2006 Ford F-150 with 4 wheels, a 6.0ltr e
```

Figure 40: the GUI with all the new features implemented

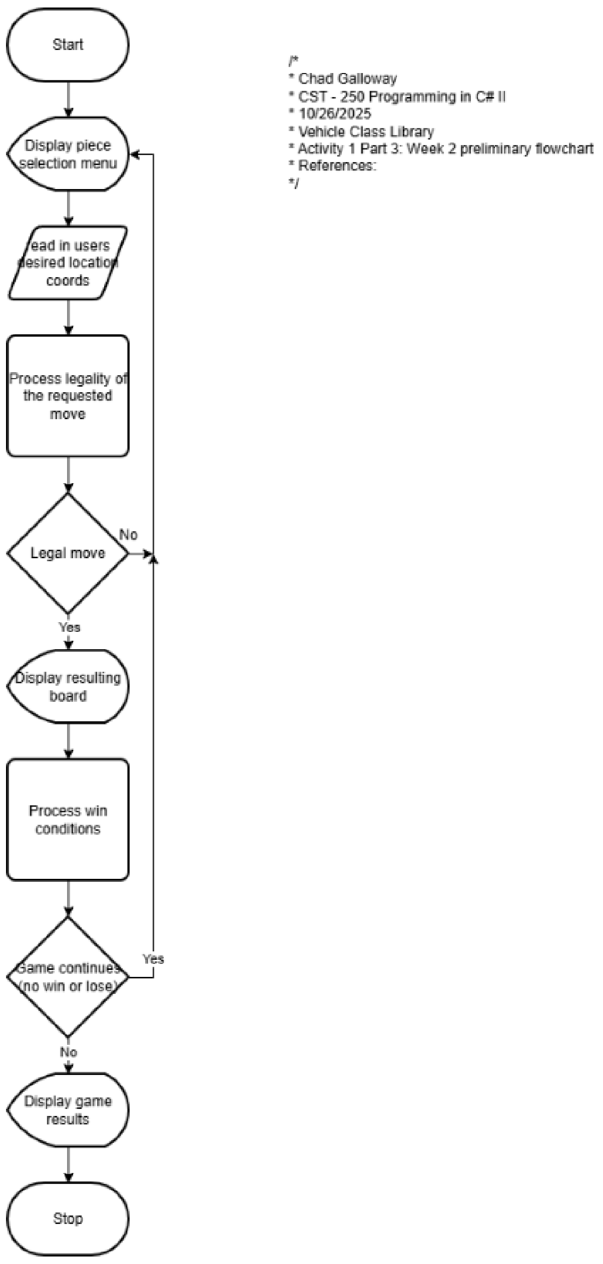


```
Car, Frod, Bronco, Blue, 2020, 62000, 4, 3.7, True, 2.2
Pickup, Ford, F-150, Blue, 2006, 28000, 4, 6.0, False, 38.6
Car, Honda, Odessey, Gray, 2015, 35000, 4, 3.5, False, 8.5
```

Figure 41: The contents of the save file after hitting save inventory in the above figure 40 screenshot

Part 3

Flowchart


Start
Display piece selection menu
ead in users esired locatio coords
Process legality o the requested move
Legal move
No
Yes
Display resulting board
Process win conditions
Game continues (no win or lose)
Yes
No
Display game results
Stop

```
/*
* Chad Galloway
* CST - 250 Programming in C# II
* 10/26/2025
* Vehicle Class Library
* Activity 1 Part 3: Week 2 preliminary flowchart
* References:
*/
```

Figure 42: Screenshot Flowchart for week 2

UML

```
/*
 * Chad Galloway
 * CST - 250 Programming in C# II
 * 10/26/2025
 * Vehicle Class Library
 * Activity 1 Part 3: Week 2 UML
 * References:
 */
```

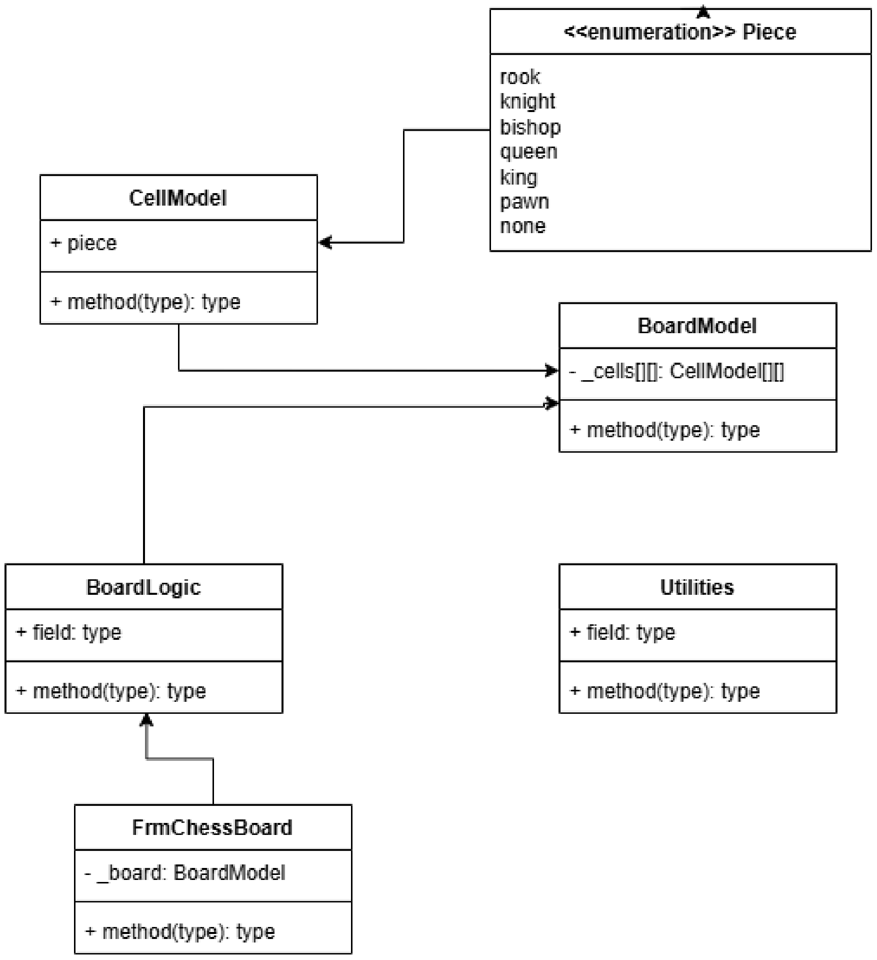


Figure 43: Screenshot of week 2 UML