

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

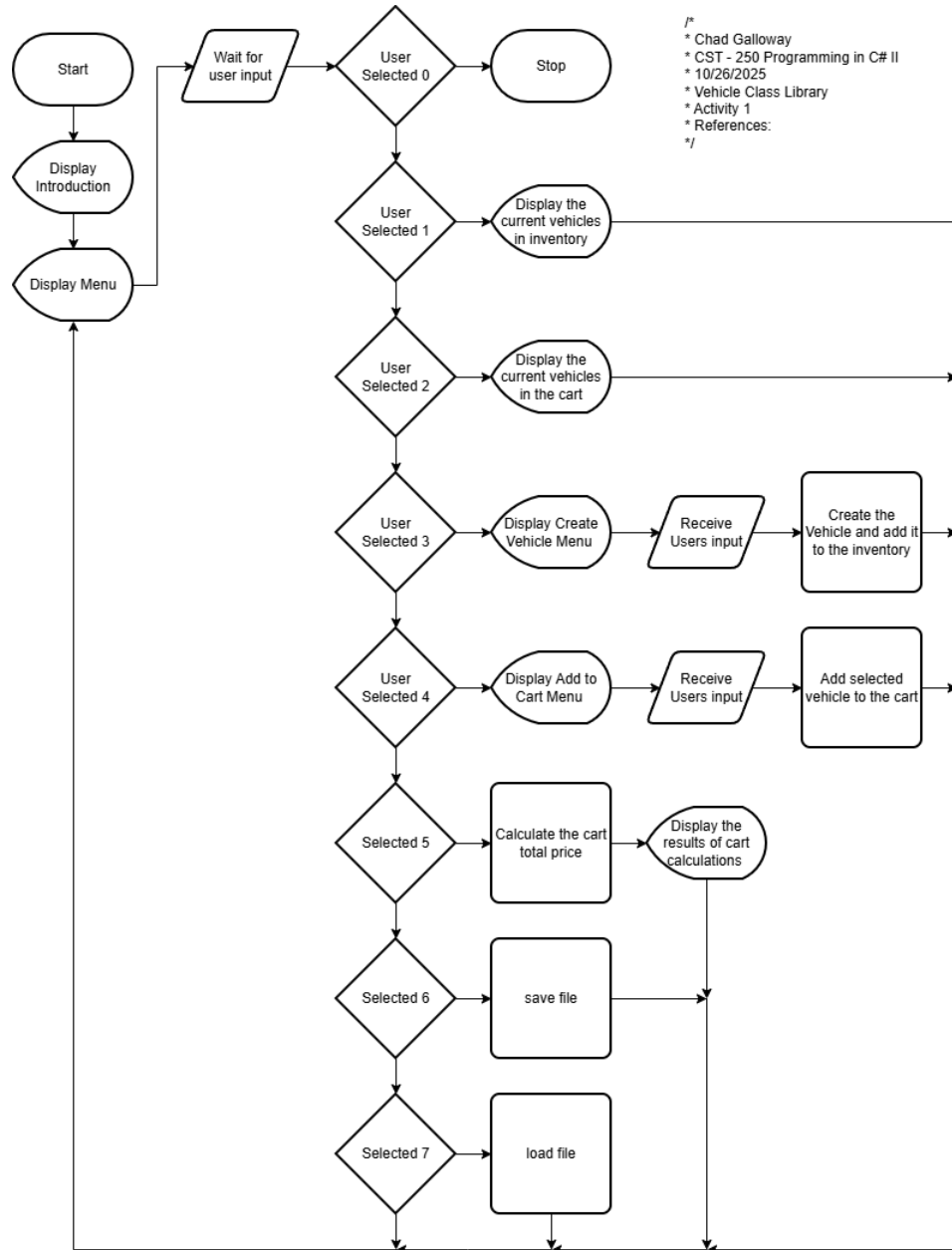


Figure X: Flow chart of Activity 1

UML Class Diagram

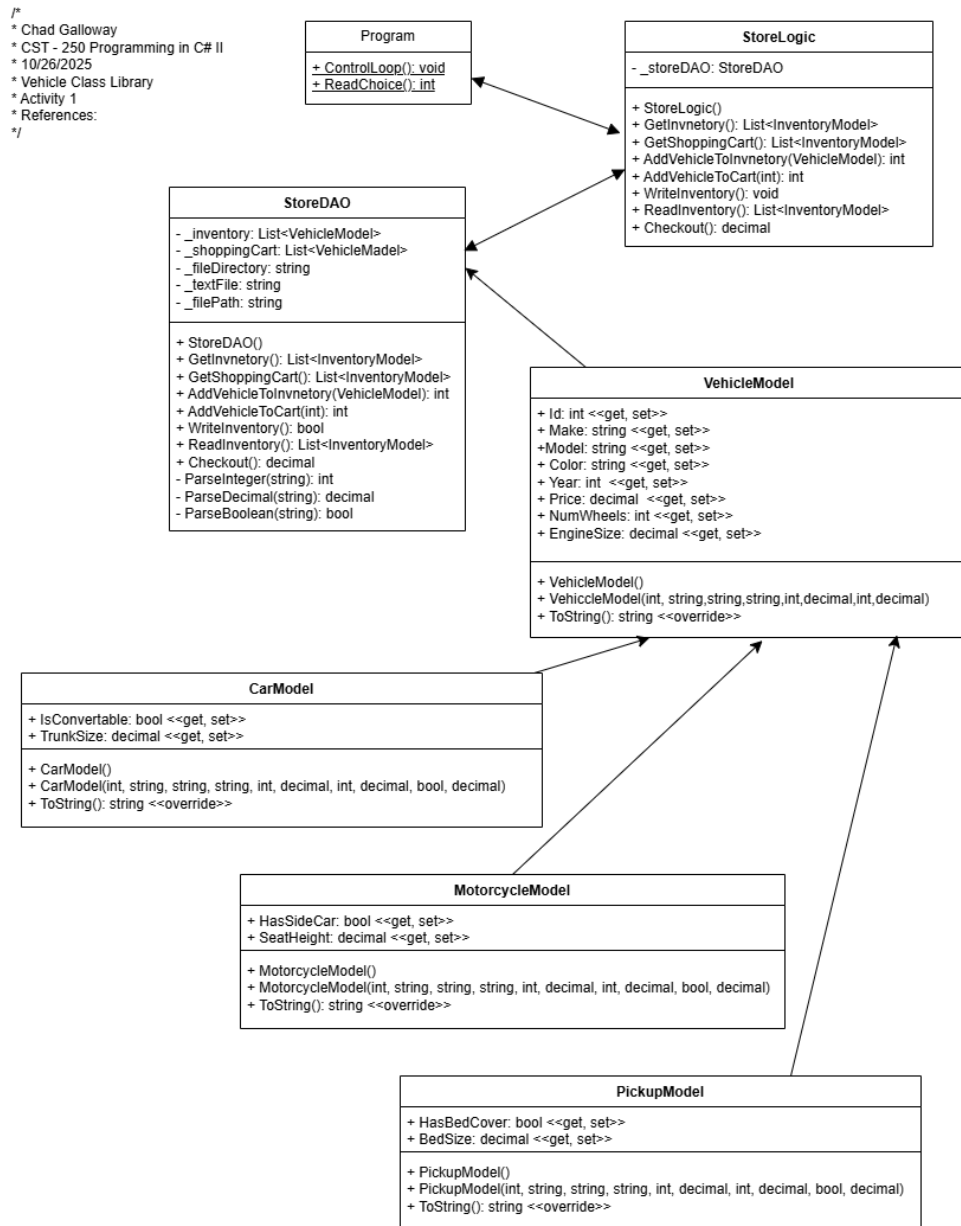


Figure X: UML Class Diagram

Screen Shots

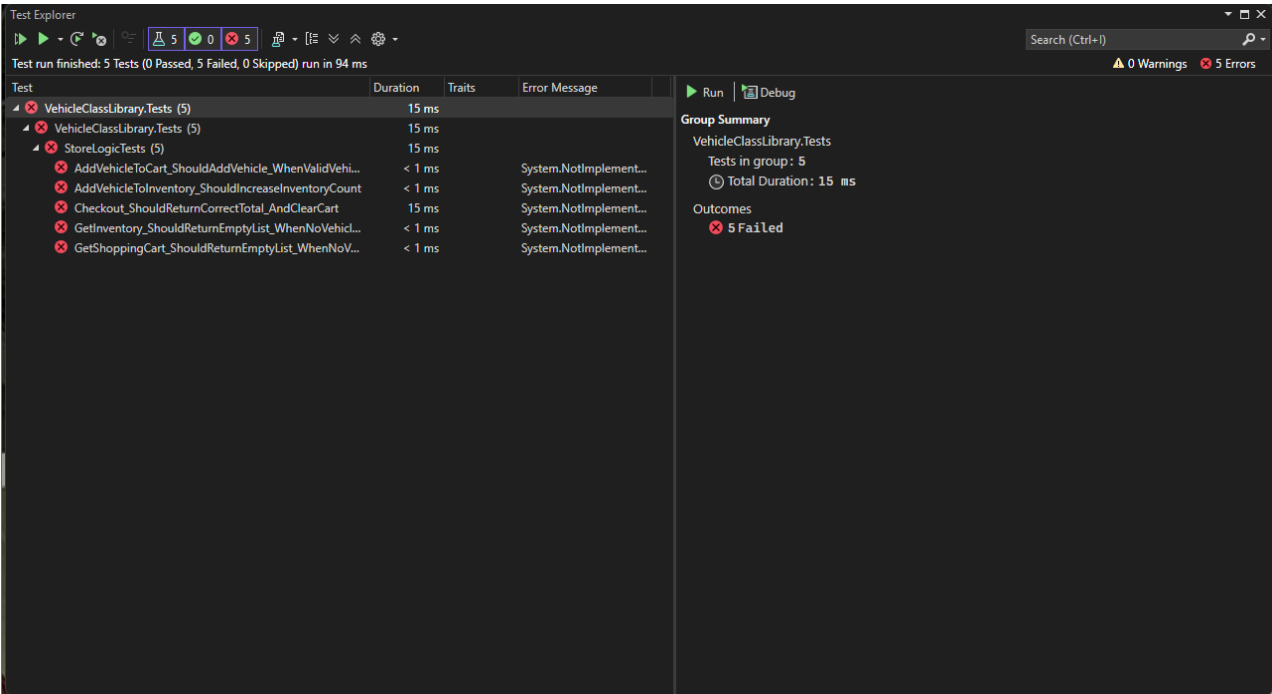


Figure 3: Screenshot of xUnit test

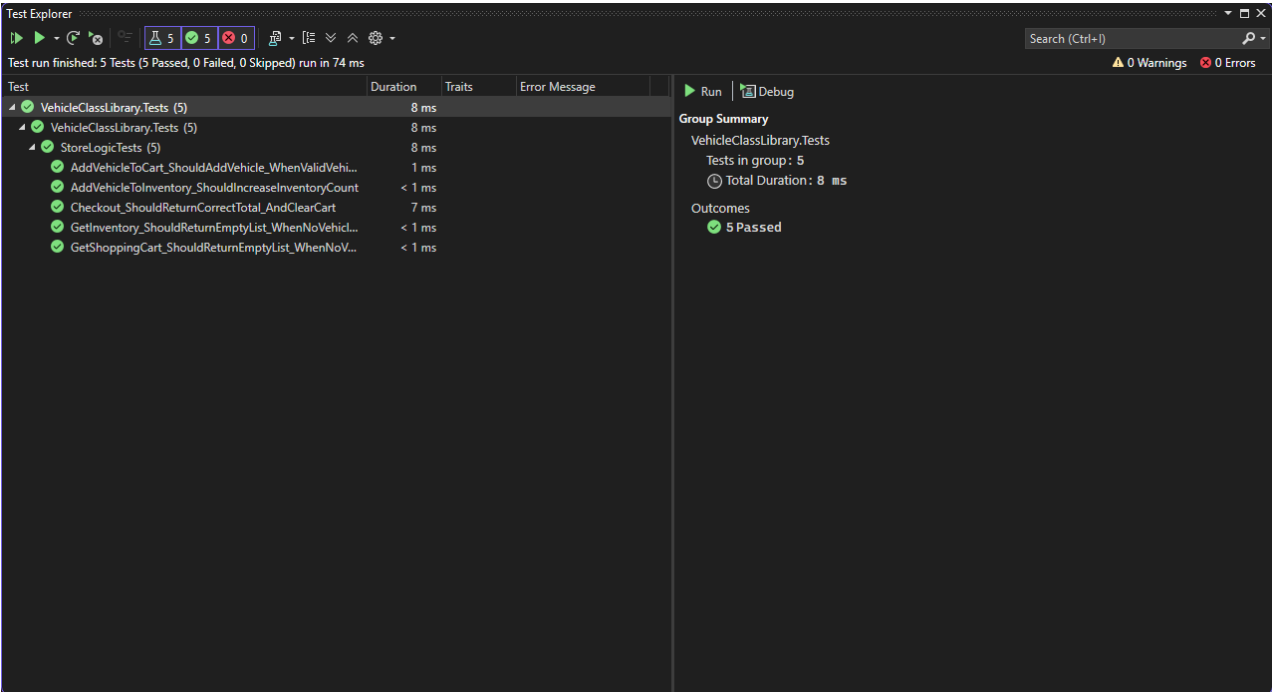


Figure 4: Screenshot of unit test

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





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.

```

VehicleModels.cs  CarModels.cs  MotorcycleModels.cs  PickupModels.cs  StoreDAO.cs  StoreDAO.cs
VehicleClassLibrary.Services.DataAccessLayer.StoreDAO

1  // Chad Gallaway
2  // C# - 200 Programming in C# 11
3  // 10/26/2020
4  // Vehicle Class Library
5  // Activity 3
6  // References:
7  //
8
9  using VehicleClassLibrary.Models;
10
11 namespace VehicleClassLibrary.Services.DataAccessLayer
12 {
13     // Summary:
14     // public class StoreDAO
15     {
16         // CarModel list for the stores inventory
17         private List<VehicleModel> _inventory;
18         // CarModel list for the users shopping cart
19         private List<VehicleModel> _shoppingCart;
20         // The directory for the inventory test file
21         private string _filePath;
22         // The name of the inventory test file
23         private string _testFile = "inventory.txt";
24         // The full path to the test file
25         private string _filePath;
26
27         /// <summary>
28         /// Default constructor for StoreDAO
29         /// </summary>
30         public StoreDAO()
31         {
32             // Initialize the vehicle model list
33             _inventory = new List<VehicleModel>();
34             _shoppingCart = new List<VehicleModel>();
35             // Set up the file to the inventory test file
36             _filePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, _filePath, _testFile);
37         }
38
39         /// <summary>
40         /// Get a list of vehicles in the inventory
41         /// </summary>
42         public List<VehicleModel> GetInventory()
43         {
44             // Return the inventory list
45             return _inventory;
46         }
47
48         /// <summary>
49         /// Get a list of the vehicles in the users shopping cart
50         /// </summary>
51         public List<VehicleModel> GetShoppingCart()
52         {
53             // Return the shopping cart list
54             return _shoppingCart;
55         }
56
57         /// <summary>
58         /// Add a new vehicle to the inventory
59         /// </summary>
60         /// <param name="vehicle">vehicle</param>
61         /// <returns>vehicle</returns>
62         public int AddVehicleToInventory(VehicleModel vehicle)
63         {
64             // Set the id for the new vehicle
65             vehicle.Id = _inventory.Count + 1;
66             // Add the vehicle to the inventory list
67             _inventory.Add(vehicle);
68             // Return the id of the new vehicle
69             return vehicle.Id;
70         }
71
72         /// <summary>
73         /// Add a vehicle to the shopping cart based on the vehicle id
74         /// </summary>
75         /// <param name="vehicleId">vehicleId</param>
76         /// <returns>vehicle</returns>
77         // Summary:
78     }
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 9: StoreDAO citations and constructors.

```

VehicleModels CarModels MotorcycleModels PickupModels StoreDAOcs X Stereologics
VehicleClassLibrary - VehicleClassLibrary.Services.DataAccessLayer.StoreDAO - @_Inventory

12 // summary
13 // Add a vehicle to the shopping cart based on the vehicle id
14 // @param
15 // @return
16 // returns
17
18 public int AddVehicleToCart(int vehicleId)
19 {
20     // Loop through the inventory to find the correct vehicle
21     for (int i = 0; i < _inventory.Count; i++)
22     {
23         // Check if the inventory vehicle id matches the parameters
24         if (_inventory[i].Id == vehicleId)
25         {
26             // If so, add the vehicle to the shopping cart
27             _shoppingCart.Add(_inventory[i]);
28         }
29     }
30     // Return the number of vehicles in the shopping cart
31     return _shoppingCart.Count;
32 }
33
34 // summary
35 // Write the inventory to a text file
36 // @param
37 // @return
38 // returns
39
40 public bool WriteInventory()
41 {
42     // Check if the directory exists
43     if (!Directory.Exists(_filePath))
44     {
45         // If the directory does not exist, create it
46         Directory.CreateDirectory(_filePath);
47     }
48
49     // Try/Catch any exceptions with the stream writer
50     try
51     {
52         // Create the stream writer to write to the file
53         StreamWriter writer = new StreamWriter(_filePath);
54
55         // Loop through each vehicle in the inventory
56         foreach (VehicleModel vehicle in _inventory)
57         {
58             type vehicleType = vehicle.GetType();
59             switch (vehicleType.Name)
60             {
61                 case "CarModel":
62                     // Cast the vehicle to a car
63                     CarModel car = (CarModel)vehicle;
64                     // Write the car to the file
65                     writer.WriteLine($"{car.Model}, {car.Color}, {car.Year}, {car.Price}, {car.NumWheels}, {car.EngineSize}, {car.IsConvertible}, {car.TrunkSize}");
66                     break;
67
68                 case "MotorcycleModel":
69                     // Cast the vehicle to a motorcycle
70                     MotorcycleModel motorcycle = (MotorcycleModel)vehicle;
71                     // Write the motorcycle to the file
72                     writer.WriteLine($"{motorcycle.Make}, {motorcycle.Model}, {motorcycle.Color}, {motorcycle.Year}, {motorcycle.Price}, {motorcycle.NumWheels}, {motorcycle.EngineSize}, {motorcycle.HasSideCar}, {motorcycle.SeatHeight}");
73                     break;
74
75                 case "PickupModel":
76                     // Cast the vehicle to a pickup
77                     PickupModel pickup = (PickupModel)vehicle;
78                     // Write the pickup to the file
79                     writer.WriteLine($"{pickup.Make}, {pickup.Model}, {pickup.Color}, {pickup.Year}, {pickup.Price}, {pickup.NumWheels}, {pickup.EngineSize}, {pickup.HasBedCover}, {pickup.BedSize}");
80                     break;
81
82                 default:
83                     // Write the vehicle to the file
84                     writer.WriteLine($"{vehicle.Make}, {vehicle.Model}, {vehicle.Color}, {vehicle.Year}, {vehicle.Price}, {vehicle.NumWheels}, {vehicle.EngineSize}");
85                     break;
86             }
87         }
88     }
89     // Return if all the data was saved to the file
90     return true;
91 }
92 catch (Exception)
93 {
94 }
95 }

```

Figure 10: continuation of StoreDAO

```

VehicleModels CarModels MotorcycleModels PickupModels StoreDAOcs X Stereologics
VehicleClassLibrary - VehicleClassLibrary.Services.DataAccessLayer.StoreDAO - AddVehicleToCart(int vehicleId)

100 catch (Exception)
101 {
102     // Return false if an exception was thrown
103     return false;
104 }
105 // End of WriteInventory method
106
107 // summary
108 // Read the inventory from a text file
109 // @param
110 // @return
111 // returns
112
113 public List<VehicleModel> ReadInventory()
114 {
115     // Declare and initialize
116     string line = "";
117     string[] parts = {};
118     string make = "", model = "", color = "";
119     int year = 0, numWheels = 0;
120     decimal price = 0m, engineSize = 0m;
121     // declare vehicle variables
122     bool isConvertible = false, hasSideCar = false, hasBedCover = false;
123     decimal trunkSize = 0m, seatHeight = 0m, bedSize = 0m;
124
125     // try/catch any exceptions with the stream reader
126     try
127     {
128         // Check if the file exists
129         if (File.Exists(_filePath))
130         {
131             // Create the stream reader to read from the file
132             using (StreamReader reader = new StreamReader(_filePath))
133             {
134                 while ((line = reader.ReadLine()) != null)
135                 {
136                     // Split the line into parts on a comma-space
137                     // Create a string array to put all the separate vehicle parts into
138                     parts = line.Split(", ");
139
140                     // Use the parts array to get the common data (make, model, year, price, numWheels)
141                     make = parts[0];
142                     model = parts[1];
143                     color = parts[2];
144                     // Parse the year of the vehicle
145                     year = ParseInteger(parts[3]);
146                     // Parse the price of the vehicle
147                     price = ParseDecimal(parts[4]);
148                     // Parse the number of wheels
149                     numWheels = ParseInteger(parts[5]);
150                     // Parse the engine size
151                     engineSize = ParseDecimal(parts[6]);
152
153                     // Use the first part of the data to create a switch for the specific model
154                     switch (parts[0])
155                     {
156                         case "Car":
157                             // Parse the convertible status for the car
158                             isConvertible = ParseBoolean(parts[7]);
159                             // Parse the trunk size for the car
160                             trunkSize = ParseDecimal(parts[8]);
161                             // Create a new car model and add it to the inventory
162                             CarModel car = new CarModel(0, make, model, color, year, price, numWheels, engineSize, isConvertible, trunkSize);
163                             // Add the car to the inventory
164                             AddVehicleToInventory(car);
165                             break;
166
167                         case "Motorcycle":
168                             // Parse the side car status for the motorcycle
169                             hasSideCar = ParseBoolean(parts[7]);
170                             // Parse the seat height for the motorcycle
171                             seatHeight = ParseDecimal(parts[8]);
172                             // Create a new motorcycle using the read properties
173                             MotorcycleModel motorcycle = new MotorcycleModel(0, make, model, color, year, price, numWheels, engineSize, hasSideCar, seatHeight);
174                             // Add the motorcycle to the inventory
175                             AddVehicleToInventory(motorcycle);
176                             break;
177
178                         case "Pickup":
179                             // Parse the bed cover status for the pickup
180                             hasBedCover = ParseBoolean(parts[7]);
181                             // Parse the bed size for the pickup
182

```

Figure 11: continuation of StoreDAO


```

221         break;
222
223     case "Pickup":
224         // Parse the bed cover status for the pickup
225         hasBedCover = ParseBoolean(parts[1]);
226         // Parse the bed size on the pickup
227         bedSize = ParseDecimal(parts[2]);
228         // Create a new pickup using the read properties
229         PickupModel pickup = new PickupModel(0, make, model, color, year, price, numWheels, engineSize, hasBedCover, bedSize);
230         AddVehicleInventory(pickup);
231         break;
232
233     default:
234         // Create a new vehicle using the read properties
235         VehicleModel vehicle = new VehicleModel(0, make, model, color, year, price, numWheels, engineSize);
236         AddVehicleInventory(vehicle);
237         break;
238     }
239 }
240
241 // End of AddVehicleInventory method
242
243 // Return the inventory list as is
244 return _inventory;
245 }
246
247 // Return the inventory list
248 return _inventory;
249 }
250
251 // End of ReadInventory method
252
253 /// Summary
254 /// Method to safely parse an integer
255 /// Summary
256 /// param Name="input"~/param
257 /// returns~/returns
258
259 private int ParseInteger(string input)
260 {
261     try
262     {
263         // Parse the input and return
264         return int.Parse(input);
265     }
266     catch (Exception ex)
267     {
268         // Return 0
269         return 0;
270     }
271 }
272
273 /// Summary
274 /// Method to safely parse a decimal
275 /// Summary
276 /// param Name="input"~/param
277 /// returns~/returns
278
279 private decimal ParseDecimal(string input)
280 {
281     try
282     {
283         // Parse the input and return
284         return decimal.Parse(input);
285     }
286     catch (Exception ex)
287     {
288         // Return 0
289         return 0;
290     }
291 }
292
293 /// Summary
294 /// Method to safely parse a boolean
295 /// Summary
296 /// param Name="input"~/param
297 /// returns~/returns
298
299 private bool ParseBoolean(string input)
300 {
301     try
302     {
303         // Parse the input and return
304         return bool.Parse(input);
305     }
306     catch (Exception ex)
307     {
308         // Return false
309         return false;
310     }
311 }
312
313 /// Summary
314 /// Get the total of the users shopping cart and clear the cart
315 /// Summary
316 /// returns~/returns
317
318 public decimal Checkout()
319 {
320     // Set up a variable to keep track of the carts total
321     decimal total = 0;
322     // Loop through each vehicle in the shopping cart
323     foreach (VehicleModel vehicle in _shoppingCart)
324     {
325         // Add the vehicle price to the total variable
326         total += vehicle.Price;
327     }
328     // Clear the cart
329     _shoppingCart.Clear();
330     // Return the total
331     return total;
332 }
333

```

Figure 12: continuation of StoreDAO

```

301     try
302     {
303         // Parse the input and return
304         return int.Parse(input);
305     }
306     catch (Exception ex)
307     {
308         // Return 0
309         return 0;
310     }
311 }
312
313 /// Summary
314 /// Method to safely parse a decimal
315 /// Summary
316 /// param Name="input"~/param
317 /// returns~/returns
318
319 private decimal ParseDecimal(string input)
320 {
321     try
322     {
323         // Parse the input and return
324         return decimal.Parse(input);
325     }
326     catch (Exception ex)
327     {
328         // Return 0
329         return 0;
330     }
331 }
332
333 /// Summary
334 /// Method to safely parse a boolean
335 /// Summary
336 /// param Name="input"~/param
337 /// returns~/returns
338
339 private bool ParseBoolean(string input)
340 {
341     try
342     {
343         // Parse the input and return
344         return bool.Parse(input);
345     }
346     catch (Exception ex)
347     {
348         // Return false
349         return false;
350     }
351 }
352
353 /// Summary
354 /// Get the total of the users shopping cart and clear the cart
355 /// Summary
356 /// returns~/returns
357
358 public decimal Checkout()
359 {
360     // Set up a variable to keep track of the carts total
361     decimal total = 0;
362     // Loop through each vehicle in the shopping cart
363     foreach (VehicleModel vehicle in _shoppingCart)
364     {
365         // Add the vehicle price to the total variable
366         total += vehicle.Price;
367     }
368     // Clear the cart
369     _shoppingCart.Clear();
370     // Return the total
371     return total;
372 }
373

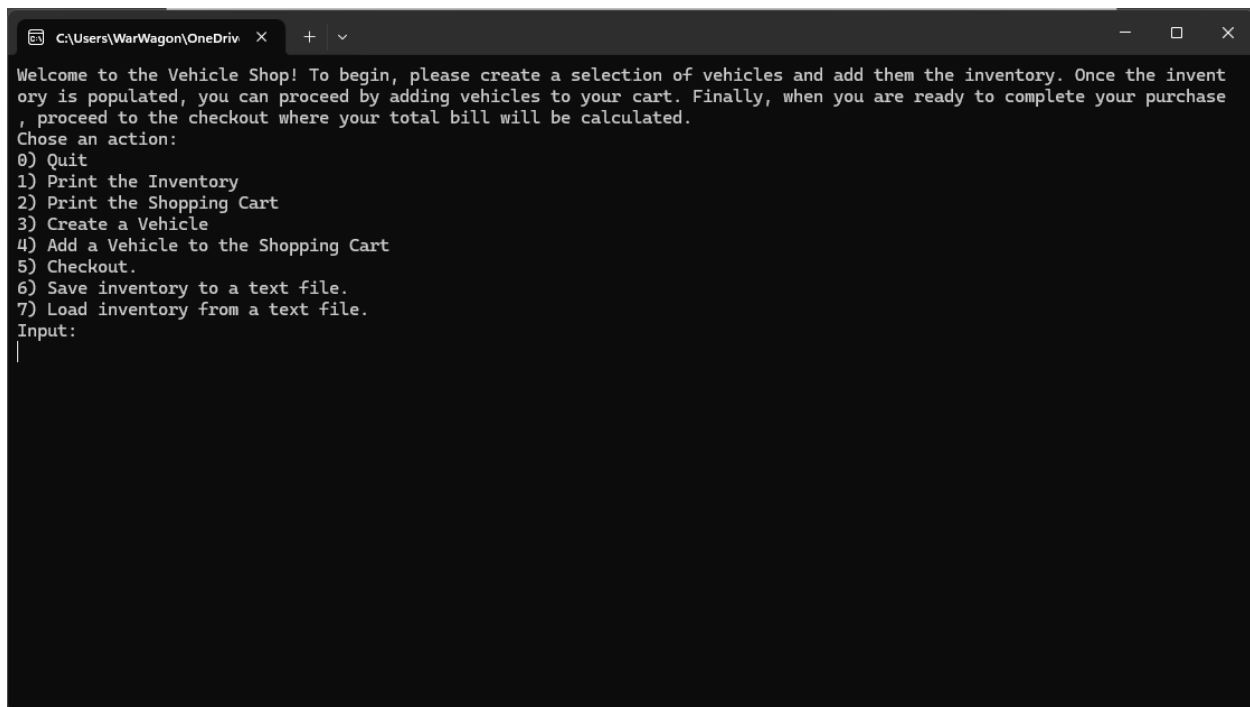
```

Figure 13: Final screenshot of StoreDAO

Figure 14 and 15 are screenshots of the StoreLogic class.

Figure 14: StoreLogic class citations and constructor

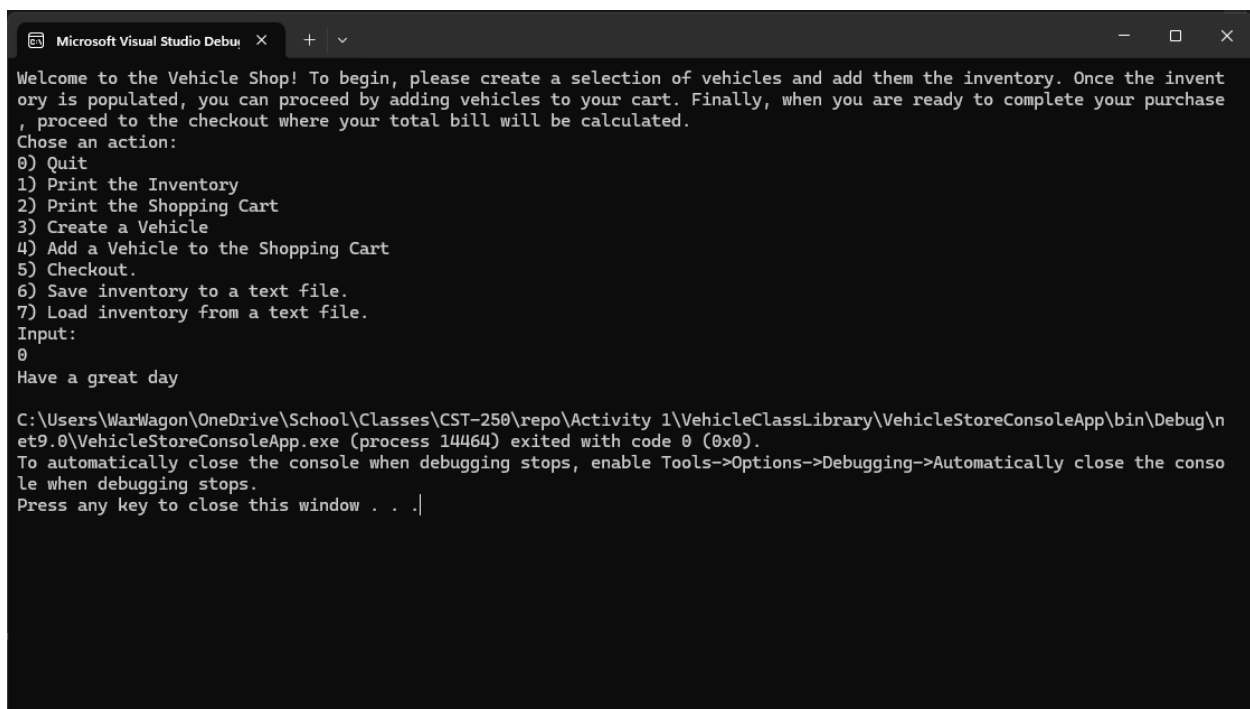
Figure 15: StoreLogic methods



A screenshot of a console application window. The title bar shows the file path 'C:\Users\WarWagon\OneDrive'. The text inside the window reads: 'Welcome to the Vehicle Shop! To begin, please create a selection of vehicles and add them the inventory. Once the inventory 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:'. The cursor is positioned at the end of the 'Input:' line.

```
C:\Users\WarWagon\OneDrive
Welcome to the Vehicle Shop! To begin, please create a selection of vehicles and add them the inventory. Once the inventory 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



A screenshot of a console application window, likely running in a debugger. The title bar shows 'Microsoft Visual Studio Debug'. The text inside the window is identical to Figure 16, but the 'Input:' line now shows '0'. Below the menu, the text 'Have a great day' is displayed. At the bottom, a message states: 'C:\Users\WarWagon\OneDrive\School\Classes\CST-250\repo\Activity 1\VehicleClassLibrary\VehicleStoreConsoleApp\bin\Debug\net9.0\VehicleStoreConsoleApp.exe (process 14464) exited with code 0 (0x0). To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops. Press any key to close this window . . .|'. The cursor is at the end of the last line.

```
Microsoft Visual Studio Debug
Welcome to the Vehicle Shop! To begin, please create a selection of vehicles and add them the inventory. Once the inventory 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\net9.0\VehicleStoreConsoleApp.exe (process 14464) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|
```

Figure 17: user input 0

```
C:\Users\WarWagon\OneDrive  X + v
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

```
C:\Users\WarWagon\OneDrive  X + v
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

```
C:\Users\WarWagon\OneDrive >
Welcome to the Vehicle Shop! To begin, please create a selection of vehicles and add them the inventory. Once the inventory 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

```
C:\Users\WarWagon\OneDrive >
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

```
C:\Users\WarWagon\OneDrive  X + v
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

```
C:\Users\WarWagon\OneDrive  X + v
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

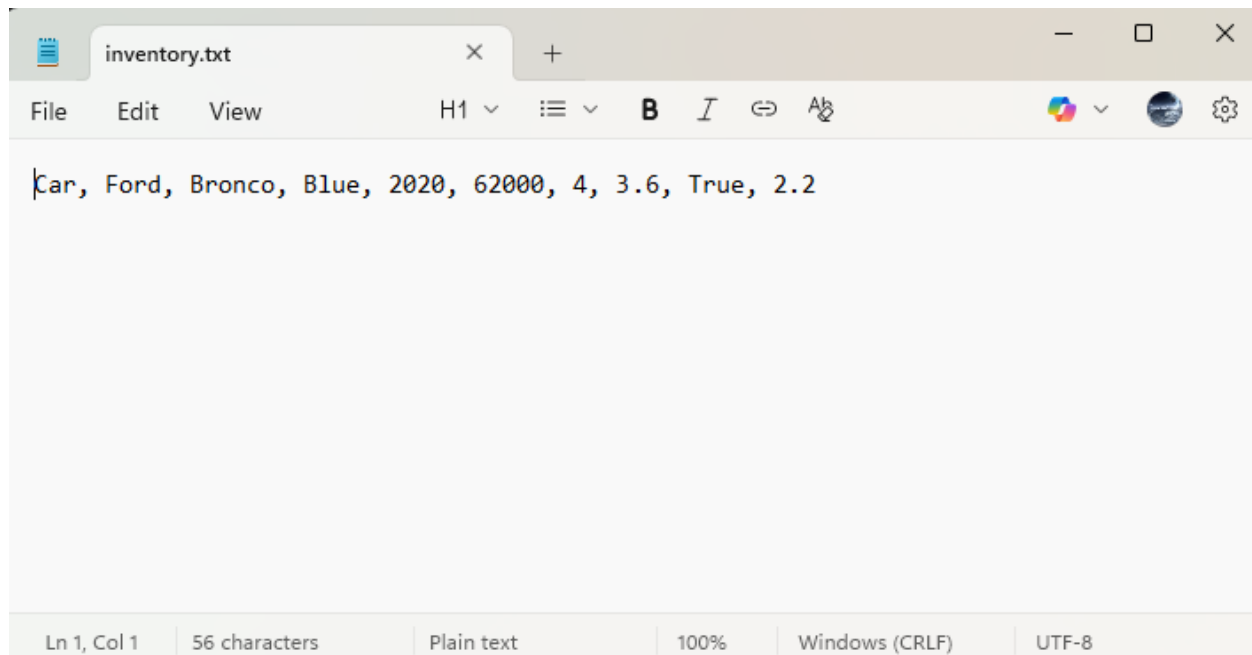


Figure 24: File contents after save (number 6)

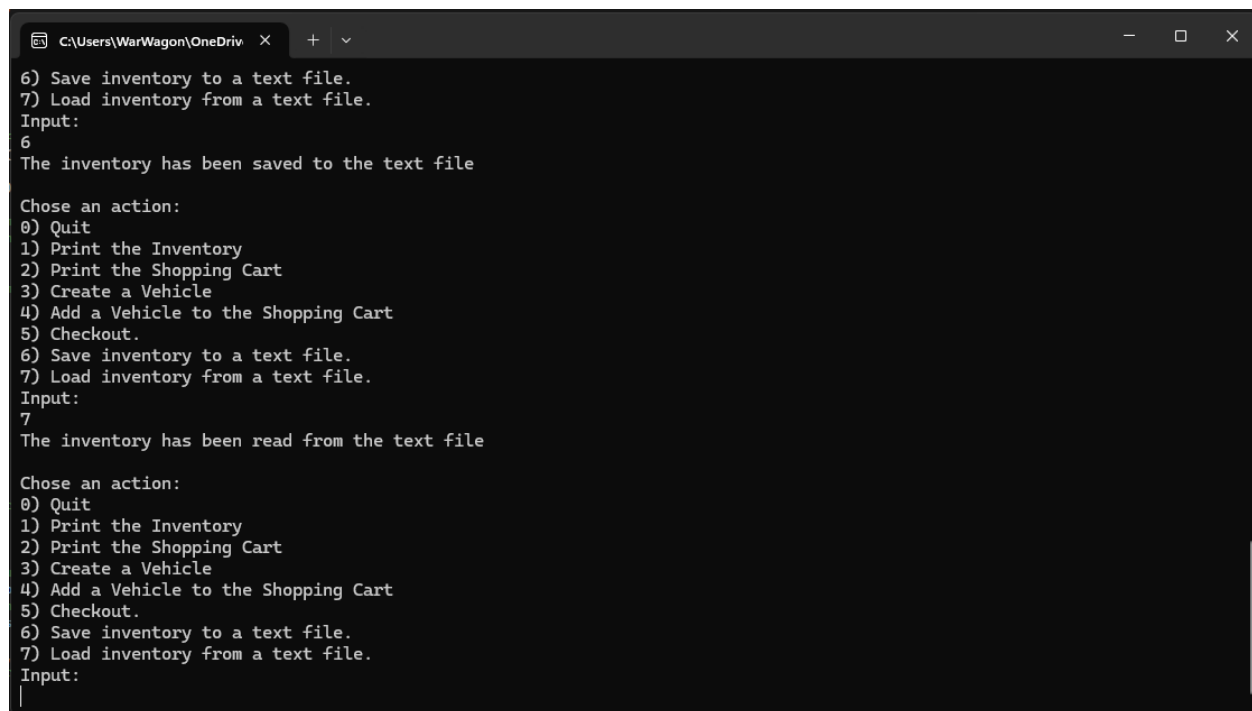


Figure 25: user input 7

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

```

100 // Write the inventory to a text file
101 // Save inventory
102 // Return true
103 public bool WriteInventory()
104 {
105     // Check if the directory exists
106     if (Directory.Exists(_filePath))
107     {
108         // If the directory does not exist, create it
109         Directory.CreateDirectory(_filePath);
110     }
111
112     // Try/Catch any exceptions with the stream writer
113     try
114     {
115         // Create the stream writer to write to the file
116         using StreamWriter writer = new StreamWriter(_filePath);
117
118         // Loop through each vehicle in the inventory
119         foreach (Vehicle vehicle in _inventory)
120         {
121             type vehicleType = vehicle.GetType();
122             switch (vehicleType.Name)
123             {
124                 case "CarModel":
125                     // Create the car object
126                     CarModel car = new CarModel();
127                     // Write the car to the file
128                     writer.WriteLine($"{car.Model}, {car.Color}, {car.Year}, {car.Price}, {car.NumWheels}, {car.EngineSize}, {car.IsConvertible}, {car.TrunkSize}");
129                     break;
130
131                 case "MotorcycleModel":
132                     // Create the motorcycle object
133                     MotorcycleModel motorcycle = new MotorcycleModel();
134                     // Write the motorcycle to the file
135                     writer.WriteLine($"{motorcycle.Model}, {motorcycle.Color}, {motorcycle.Year}, {motorcycle.Price}, {motorcycle.NumWheels}, {motorcycle.EngineSize}, {motorcycle.IsConvertible}, {motorcycle.TrunkSize}");
136                     break;
137
138                 case "PickupModel":
139                     // Create the pickup object
140                     PickupModel pickup = new PickupModel();
141                     // Write the pickup to the file
142                     writer.WriteLine($"{pickup.Model}, {pickup.Color}, {pickup.Year}, {pickup.Price}, {pickup.NumWheels}, {pickup.EngineSize}, {pickup.IsConvertible}, {pickup.TrunkSize}");
143                     break;
144
145                 default:
146                     // Write the vehicle to the file
147                     writer.WriteLine($"{vehicle.Model}, {vehicle.Color}, {vehicle.Year}, {vehicle.Price}, {vehicle.NumWheels}, {vehicle.EngineSize}");
148                     break;
149             }
150         }
151     }
152     // Return if all the data was saved to the file
153     return true;
154 }
155
156 // Save inventory
157 // Save inventory
158 // Return true
159 public List<VehicleModel> ReadInventory()
160 {
161     // Return and initialize
162     string line = "";
163     string[] parts = {};
164     string name = "", model = "", color = "";
165     int year = 0, numWheels = 0;
166     decimal price = 0, engineSize = 0;
167     // Create the vehicle object
168     bool isConvertible = false, hasTrunk = false, hasTrunkSize = false;
169     decimal trunkSize = 0, numWheels = 0, engineSize = 0;
170
171     // Try/Catch any exceptions with the stream reader
172     try
173     {
174         // Check if the file exists
175         if (File.Exists(_filePath))
176         {
177             // Create the stream reader to read from the file
178             using StreamReader reader = new StreamReader(_filePath);
179
180             while ((line = reader.ReadLine()) != null)
181             {
182                 // Split the line into parts as a comma-separated
183                 // Create a string array to get all the individual vehicle parts data
184                 parts = line.Split(",");
185
186                 // Use the parts array to get the common data (name, model, year, price, numWheels)
187                 name = parts[0];
188                 model = parts[1];
189                 year = parts[2];
190                 price = parts[3];
191                 numWheels = parts[4];
192                 engineSize = parts[5];
193
194                 // Use the parts of the data to create a switch for the specific model
195                 switch (parts[0])
196                 {
197                     case "Car":
198                         // Create the car object
199                         CarModel car = new CarModel();
200                         // Write the car to the inventory
201                         AddVehicleToInventory(car);
202                         break;
203
204                     case "Motorcycle":
205                         // Create the motorcycle object
206                         MotorcycleModel motorcycle = new MotorcycleModel();
207                         // Write the motorcycle to the inventory
208                         AddVehicleToInventory(motorcycle);
209                         break;
210
211                     case "Pickup":
212                         // Create the pickup object
213                         PickupModel pickup = new PickupModel();
214                         // Write the pickup to the inventory
215                         AddVehicleToInventory(pickup);
216                         break;
217
218                     default:
219                         // Create a new vehicle using the read properties
220                         VehicleModel vehicle = new VehicleModel();
221                         AddVehicleToInventory(vehicle);
222                         break;
223                 }
224             }
225         }
226     }
227 }

```

Figure 26: Challenge Changes

```

100 // Write the inventory to a text file
101 // Save inventory
102 // Return true
103 public bool WriteInventory()
104 {
105     // Check if the directory exists
106     if (Directory.Exists(_filePath))
107     {
108         // If the directory does not exist, create it
109         Directory.CreateDirectory(_filePath);
110     }
111
112     // Try/Catch any exceptions with the stream writer
113     try
114     {
115         // Create the stream writer to write to the file
116         using StreamWriter writer = new StreamWriter(_filePath);
117
118         // Loop through each vehicle in the inventory
119         foreach (Vehicle vehicle in _inventory)
120         {
121             type vehicleType = vehicle.GetType();
122             switch (vehicleType.Name)
123             {
124                 case "CarModel":
125                     // Create the car object
126                     CarModel car = new CarModel();
127                     // Write the car to the file
128                     writer.WriteLine($"{car.Model}, {car.Color}, {car.Year}, {car.Price}, {car.NumWheels}, {car.EngineSize}, {car.IsConvertible}, {car.TrunkSize}");
129                     break;
130
131                 case "MotorcycleModel":
132                     // Create the motorcycle object
133                     MotorcycleModel motorcycle = new MotorcycleModel();
134                     // Write the motorcycle to the file
135                     writer.WriteLine($"{motorcycle.Model}, {motorcycle.Color}, {motorcycle.Year}, {motorcycle.Price}, {motorcycle.NumWheels}, {motorcycle.EngineSize}, {motorcycle.IsConvertible}, {motorcycle.TrunkSize}");
136                     break;
137
138                 case "PickupModel":
139                     // Create the pickup object
140                     PickupModel pickup = new PickupModel();
141                     // Write the pickup to the file
142                     writer.WriteLine($"{pickup.Model}, {pickup.Color}, {pickup.Year}, {pickup.Price}, {pickup.NumWheels}, {pickup.EngineSize}, {pickup.IsConvertible}, {pickup.TrunkSize}");
143                     break;
144
145                 default:
146                     // Write the vehicle to the file
147                     writer.WriteLine($"{vehicle.Model}, {vehicle.Color}, {vehicle.Year}, {vehicle.Price}, {vehicle.NumWheels}, {vehicle.EngineSize}");
148                     break;
149             }
150         }
151     }
152     // Return if all the data was saved to the file
153     return true;
154 }
155
156 // Save inventory
157 // Save inventory
158 // Return true
159 public List<VehicleModel> ReadInventory()
160 {
161     // Return and initialize
162     string line = "";
163     string[] parts = {};
164     string name = "", model = "", color = "";
165     int year = 0, numWheels = 0;
166     decimal price = 0, engineSize = 0;
167     // Create the vehicle object
168     bool isConvertible = false, hasTrunk = false, hasTrunkSize = false;
169     decimal trunkSize = 0, numWheels = 0, engineSize = 0;
170
171     // Try/Catch any exceptions with the stream reader
172     try
173     {
174         // Check if the file exists
175         if (File.Exists(_filePath))
176         {
177             // Create the stream reader to read from the file
178             using StreamReader reader = new StreamReader(_filePath);
179
180             while ((line = reader.ReadLine()) != null)
181             {
182                 // Split the line into parts as a comma-separated
183                 // Create a string array to get all the individual vehicle parts data
184                 parts = line.Split(",");
185
186                 // Use the parts array to get the common data (name, model, year, price, numWheels)
187                 name = parts[0];
188                 model = parts[1];
189                 year = parts[2];
190                 price = parts[3];
191                 numWheels = parts[4];
192                 engineSize = parts[5];
193
194                 // Use the parts of the data to create a switch for the specific model
195                 switch (parts[0])
196                 {
197                     case "Car":
198                         // Create the car object
199                         CarModel car = new CarModel();
200                         // Write the car to the inventory
201                         AddVehicleToInventory(car);
202                         break;
203
204                     case "Motorcycle":
205                         // Create the motorcycle object
206                         MotorcycleModel motorcycle = new MotorcycleModel();
207                         // Write the motorcycle to the inventory
208                         AddVehicleToInventory(motorcycle);
209                         break;
210
211                     case "Pickup":
212                         // Create the pickup object
213                         PickupModel pickup = new PickupModel();
214                         // Write the pickup to the inventory
215                         AddVehicleToInventory(pickup);
216                         break;
217
218                     default:
219                         // Create a new vehicle using the read properties
220                         VehicleModel vehicle = new VehicleModel();
221                         AddVehicleToInventory(vehicle);
222                         break;
223                 }
224             }
225         }
226     }
227 }

```

Figure 27 Challenge changes


```
C:\Users\WarWagon\OneDrive >
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

Vehicle Store

Create a Vehicle

☒ Car ☐ Motorcycle
☐ Pickup ☐ Vehicle

Make:

Model:

Color:

Year:

Price:

Wheels:

Engine Size:

Create

Specialty Properties

Is the car a convertible?
☐ Yes ☒ No

Trunk Size (cubit feet):

Inventory

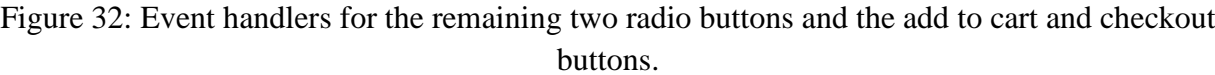
Shopping Cart

Add to Cart

Checkout

Total: \$0

Figure 29: Initial state of UI



```

frmVehicleStore.cs
VehicleStoreUIApp
frmVehicleStore
ValidateVehicleType()

// Create a new car
// Create a new motorcycle
// Create a new pickup
// Create a new vehicle
// Add the vehicle to the inventory
// Clear the fields for the next entry
// Refresh the list controls
// Display form warning label for vehicle type radio buttons
// Hide form warning label for vehicle type radio buttons

```

Figure 33: Create Button click event handler.

```

frmVehicleStore.cs
VehicleStoreUIApp
frmVehicleStore
ValidateVehicleType()

// Summary: Click event handler for the checkbox button
private void btnCheckOutClicked(object sender, EventArgs e)
{
    // Summary: Create button click event handler
    private void btnCreateClicked(object sender, EventArgs e)
    {
        // Summary: // Validation Check region //
        // Summary: //
        #region Leave event handlers for validation
        // Summary:
        private void txtNameLeave(object sender, EventArgs e)
        {
            // Validate the name textbox
            ValidateTxtName();
        }
        // Summary:
        private void txtModelLeave(object sender, EventArgs e)
        {
            // Validate the model textbox
            ValidateTxtModel();
        }
        // Summary:
        private void txtColorLeave(object sender, EventArgs e)
        {
            // Validate the color textbox
            ValidateTxtColor();
        }
        // Summary:
        private void txtYearLeave(object sender, EventArgs e)
        {
            // Validate the year textbox
            ValidateTxtYear();
        }
        // Summary:
        private void txtPriceLeave(object sender, EventArgs e)
        {
            // Validate the price textbox
            ValidateTxtPrice();
        }
        // Summary:
        private void txtWheelsLeave(object sender, EventArgs e)
        {
            // Validate the wheels textbox
            ValidateTxtWheels();
        }
        // Summary:
        private void txtEngineLeave(object sender, EventArgs e)
        {
            // Validate the engine size textbox
            ValidateTxtEngine();
        }
        // Summary:
        private void txtSpecialLeave(object sender, EventArgs e)
        {
            // Validate the specialty decimal textbox
            ValidateTxtSpecial();
        }
        #endregion
        // Validation Check
    }
}

```

Figure 34: All the leave event handlers

```

frmVehicleStore.cs [Design]
frmVehicleStore.cs [X]
VehicleStoreCSharp.frmVehicleStore
ValidateVehicleType()

// Leave event handlers for validation
#region Validation Checks
/// <summary> Validate that the user has selected a vehicle type
private void ValidateVehicleType()
{
    if (radioCar.Checked || radioMotorcycle.Checked || radioPickup.Checked || radioWhistle.Checked)
    {
        // Hide the error label
        lblTypeWarning.Visible = false;
        // Set the flag
        isValidVehicleType = true;
    }
    else
    {
        // Show the error label
        lblTypeWarning.Visible = true;
        // Set the flag
        isValidVehicleType = false;
    }
}

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

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

/// <summary> Validate the color textbox
private string ValidateColor()
{
}

/// <summary> Validate the year textbox
private int ValidateYear()
{
}

/// <summary> Validate the price textbox
private decimal ValidatePrice()
{
}

/// <summary> Validate the wheels textbox
private int ValidateWheels()
{
}

```

Figure 35: Textbox and radio button validation checks

Part 2

```

412     }
413     // Return the test from the testbox
414     return testModel.Test;
415 }
416
417 /// <summary> Validate the color testbox
418 /// </summary>
419 private string ValidateColorTest()
420 {
421     // Test for a null/empty testbox
422     if (string.IsNullOrEmpty(testColor.Test))
423     {
424         // Set the flag
425         thisColorWarning.Visible = true;
426         isColorValid = false;
427     }
428     else
429     {
430         thisColorWarning.Visible = false;
431         // Clear the flag
432         isColorValid = true;
433     }
434     // Return the test from the testbox
435     return testColor.Test;
436 }
437
438 /// <summary> Validate the year testbox
439 /// </summary>
440 private int ValidateYearTest()
441 {
442     // <summary>
443     // Validate the price testbox
444     // </summary>
445     private decimal ValidatePriceTest()
446     {
447         // <summary> Validate the price testbox
448         // </summary>
449         private int ValidateWheelSizeTest()
450         {
451             // <summary>
452             // Validate the engine size testbox
453             // </summary>
454             private decimal ValidateEngineSizeTest()
455             {
456                 // Declares and initialize
457                 decimal engineSizeValue = -1;
458
459                 // Test for a null/empty testbox
460                 if (string.IsNullOrEmpty(testEngineSize.Test))
461                 {
462                     thisEngineSizeWarning.Visible = true;
463                     // Set the flag
464                     isEngineSizeValid = false;
465                 }
466                 else
467                 {
468                     thisEngineSizeWarning.Visible = false;
469                     // Attempt to parse the testbox value
470                     isEngineSizeValid = decimal.TryParse(testEngineSize.Test, out engineSizeValue);
471                     // If the parse failed, show the warning message
472                     if (!isEngineSizeValid)
473                     {
474                         thisEngineSizeWarning.Visible = true;
475                     }
476                 }
477                 // Return the engine
478                 return engineSizeValue;
479             }
480
481             // <summary> Validate that the user has selected a specialty boolean
482             // </summary>
483             private bool ValidateSpecialtyBooleanTest()
484             {
485                 // <summary> Validate the specialty decimal testbox
486                 // </summary>
487                 private decimal ValidateSpecialtyDecimalTest()
488                 {
489                     // <summary>
490                     // </summary>
491                     return engineSizeValue;
492                 }
493             }
494         }
495     }
496 }
497
498 //endregion
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 36: New property validation checks

```

50 // <summary>
51 // Add a new vehicle to the inventory
52 // </summary>
53 // <param name="vehicle">Vehicle</param>
54 // <returns>The id of the added vehicle or -1 for a duplicate vehicle that was not added </returns>
55 // </summary>
56 public int AddVehicleToInventory(VehicleModel vehicle)
57 {
58     // Prevent duplicate vehicle entry into the inventory
59     // Loop through the inventory and compare properties
60     for (int i = 0; i < _inventory.Count; i++)
61     {
62         if (_inventory[i].Name == vehicle.Name && _inventory[i].Model == vehicle.Model && _inventory[i].Color == vehicle.Color
63             && _inventory[i].Year == vehicle.Year && _inventory[i].Price == vehicle.Price && _inventory[i].NumWheels == vehicle.NumWheels
64             && _inventory[i].EngineSize == vehicle.EngineSize)
65         {
66             // Duplicate vehicle found
67             return -1;
68         }
69         // Check last few properties
70         if (oldCar.IsConvertible == newCar.IsConvertible && oldCar.TrunkSize == newCar.TrunkSize)
71         {
72             return -1;
73         }
74         break;
75     }
76
77     case "MotorcycleModel":
78         // Cast the vehicle and current inventory item to a car
79         MotorcycleModel newMotorcycle = (MotorcycleModel)vehicle;
80         MotorcycleModel oldMotorcycle = (MotorcycleModel)_inventory[i];
81         // Check last few properties
82         if (oldMotorcycle.HasSidecar == newMotorcycle.HasSidecar && oldMotorcycle.SeatHeight == newMotorcycle.SeatHeight)
83         {
84             return -1;
85         }
86         break;
87
88     case "PickupModel":
89         // Cast the vehicle and current inventory item to a car
90         PickupModel newPickup = (PickupModel)vehicle;
91         PickupModel oldPickup = (PickupModel)_inventory[i];
92         // Check last few properties
93         if (oldPickup.HasBedCover == newPickup.HasBedCover && oldPickup.BedSize == newPickup.BedSize)
94         {
95             return -1;
96         }
97         break;
98
99     default:
100         return -1;
101     }
102
103     // If the code reaches this point the vehicle is not a exact duplicate and we are going to add it.
104     // Set the id for the new vehicle
105     vehicle.Id = _inventory.Count + 1;
106     // Add the vehicle to the inventory list
107     _inventory.Add(vehicle);
108
109     // Return the id of the new vehicle
110     return vehicle.Id;
111 }
112
113 // <summary>
114 // Add a vehicle to the shopping cart based on the vehicle id
115 // </summary>
116 // <param name="vehicleId">VehicleId</param>
117 // <returns>The vehicle</returns>
118 public int AddVehicleToCart(int vehicleId)
119 {
120     // Loop through the inventory to find the correct vehicle
121     for (int i = 0; i < _inventory.Count; i++)
122     {
123         if (_inventory[i].Id == vehicleId)
124         {
125             // Vehicle found
126             return i;
127         }
128     }
129     // Vehicle not found
130     return -1;
131 }
132
133 // <summary>
134 // Remove a vehicle from the shopping cart based on the vehicle id
135 // </summary>
136 // <param name="vehicleId">VehicleId</param>
137 // <returns>The vehicle</returns>
138 public int RemoveVehicleFromCart(int vehicleId)
139 {
140     // Loop through the inventory to find the correct vehicle
141     for (int i = 0; i < _inventory.Count; i++)
142     {
143         if (_inventory[i].Id == vehicleId)
144         {
145             // Vehicle found
146             return i;
147         }
148     }
149     // Vehicle not found
150     return -1;
151 }
152
153 // <summary>
154 // Get the vehicle from the shopping cart based on the vehicle id
155 // </summary>
156 // <param name="vehicleId">VehicleId</param>
157 // <returns>The vehicle</returns>
158 public VehicleModel GetVehicleFromCart(int vehicleId)
159 {
160     // Loop through the inventory to find the correct vehicle
161     for (int i = 0; i < _inventory.Count; i++)
162     {
163         if (_inventory[i].Id == vehicleId)
164         {
165             // Vehicle found
166             return _inventory[i];
167         }
168     }
169     // Vehicle not found
170     return null;
171 }
172
173 // <summary>
174 // Get the vehicle from the inventory based on the vehicle id
175 // </summary>
176 // <param name="vehicleId">VehicleId</param>
177 // <returns>The vehicle</returns>
178 public VehicleModel GetVehicleFromInventory(int vehicleId)
179 {
180     // Loop through the inventory to find the correct vehicle
181     for (int i = 0; i < _inventory.Count; i++)
182     {
183         if (_inventory[i].Id == vehicleId)
184         {
185             // Vehicle found
186             return _inventory[i];
187         }
188     }
189     // Vehicle not found
190     return null;
191 }
192
193 // <summary>
194 // Get the vehicle from the inventory based on the vehicle name
195 // </summary>
196 // <param name="vehicleName">VehicleName</param>
197 // <returns>The vehicle</returns>
198 public VehicleModel GetVehicleFromInventory(string vehicleName)
199 {
200     // Loop through the inventory to find the correct vehicle
201     for (int i = 0; i < _inventory.Count; i++)
202     {
203         if (_inventory[i].Name == vehicleName)
204         {
205             // Vehicle found
206             return _inventory[i];
207         }
208     }
209     // Vehicle not found
210     return null;
211 }
212
213 // <summary>
214 // Get the vehicle from the inventory based on the vehicle model
215 // </summary>
216 // <param name="vehicleModel">VehicleModel</param>
217 // <returns>The vehicle</returns>
218 public VehicleModel GetVehicleFromInventory(string vehicleModel)
219 {
220     // Loop through the inventory to find the correct vehicle
221     for (int i = 0; i < _inventory.Count; i++)
222     {
223         if (_inventory[i].Model == vehicleModel)
224         {
225             // Vehicle found
226             return _inventory[i];
227         }
228     }
229     // Vehicle not found
230     return null;
231 }
232
233 // <summary>
234 // Get the vehicle from the inventory based on the vehicle year
235 // </summary>
236 // <param name="vehicleYear">VehicleYear</param>
237 // <returns>The vehicle</returns>
238 public VehicleModel GetVehicleFromInventory(int vehicleYear)
239 {
240     // Loop through the inventory to find the correct vehicle
241     for (int i = 0; i < _inventory.Count; i++)
242     {
243         if (_inventory[i].Year == vehicleYear)
244         {
245             // Vehicle found
246             return _inventory[i];
247         }
248     }
249     // Vehicle not found
250     return null;
251 }
252
253 // <summary>
254 // Get the vehicle from the inventory based on the vehicle price
255 // </summary>
256 // <param name="vehiclePrice">VehiclePrice</param>
257 // <returns>The vehicle</returns>
258 public VehicleModel GetVehicleFromInventory(decimal vehiclePrice)
259 {
260     // Loop through the inventory to find the correct vehicle
261     for (int i = 0; i < _inventory.Count; i++)
262     {
263         if (_inventory[i].Price == vehiclePrice)
264         {
265             // Vehicle found
266             return _inventory[i];
267         }
268     }
269     // Vehicle not found
270     return null;
271 }
272
273 // <summary>
274 // Get the vehicle from the inventory based on the vehicle numWheels
275 // </summary>
276 // <param name="vehicleNumWheels">VehicleNumWheels</param>
277 // <returns>The vehicle</returns>
278 public VehicleModel GetVehicleFromInventory(int vehicleNumWheels)
279 {
280     // Loop through the inventory to find the correct vehicle
281     for (int i = 0; i < _inventory.Count; i++)
282     {
283         if (_inventory[i].NumWheels == vehicleNumWheels)
284         {
285             // Vehicle found
286             return _inventory[i];
287         }
288     }
289     // Vehicle not found
290     return null;
291 }
292
293 // <summary>
294 // Get the vehicle from the inventory based on the vehicle engineSize
295 // </summary>
296 // <param name="vehicleEngineSize">VehicleEngineSize</param>
297 // <returns>The vehicle</returns>
298 public VehicleModel GetVehicleFromInventory(string vehicleEngineSize)
299 {
300     // Loop through the inventory to find the correct vehicle
301     for (int i = 0; i < _inventory.Count; i++)
302     {
303         if (_inventory[i].EngineSize == vehicleEngineSize)
304         {
305             // Vehicle found
306             return _inventory[i];
307         }
308     }
309     // Vehicle not found
310     return null;
311 }
312
313 // <summary>
314 // Get the vehicle from the inventory based on the vehicle isConvertible
315 // </summary>
316 // <param name="vehicleIsConvertible">VehicleIsConvertible</param>
317 // <returns>The vehicle</returns>
318 public VehicleModel GetVehicleFromInventory(bool vehicleIsConvertible)
319 {
320     // Loop through the inventory to find the correct vehicle
321     for (int i = 0; i < _inventory.Count; i++)
322     {
323         if (_inventory[i].IsConvertible == vehicleIsConvertible)
324         {
325             // Vehicle found
326             return _inventory[i];
327         }
328     }
329     // Vehicle not found
330     return null;
331 }
332
333 // <summary>
334 // Get the vehicle from the inventory based on the vehicle trunkSize
335 // </summary>
336 // <param name="vehicleTrunkSize">VehicleTrunkSize</param>
337 // <returns>The vehicle</returns>
338 public VehicleModel GetVehicleFromInventory(int vehicleTrunkSize)
339 {
340     // Loop through the inventory to find the correct vehicle
341     for (int i = 0; i < _inventory.Count; i++)
342     {
343         if (_inventory[i].TrunkSize == vehicleTrunkSize)
344         {
345             // Vehicle found
346             return _inventory[i];
347         }
348     }
349     // Vehicle not found
350     return null;
351 }
352
353 // <summary>
354 // Get the vehicle from the inventory based on the vehicle hasBedCover
355 // </summary>
356 // <param name="vehicleHasBedCover">VehicleHasBedCover</param>
357 // <returns>The vehicle</returns>
358 public VehicleModel GetVehicleFromInventory(bool vehicleHasBedCover)
359 {
360     // Loop through the inventory to find the correct vehicle
361     for (int i = 0; i < _inventory.Count; i++)
362     {
363         if (_inventory[i].HasBedCover == vehicleHasBedCover)
364         {
365             // Vehicle found
366             return _inventory[i];
367         }
368     }
369     // Vehicle not found
370     return null;
371 }
372
373 // <summary>
374 // Get the vehicle from the inventory based on the vehicle bedSize
375 // </summary>
376 // <param name="vehicleBedSize">VehicleBedSize</param>
377 // <returns>The vehicle</returns>
378 public VehicleModel GetVehicleFromInventory(string vehicleBedSize)
379 {
380     // Loop through the inventory to find the correct vehicle
381     for (int i = 0; i < _inventory.Count; i++)
382     {
383         if (_inventory[i].BedSize == vehicleBedSize)
384         {
385             // Vehicle found
386             return _inventory[i];
387         }
388     }
389     // Vehicle not found
390     return null;
391 }
392
393 // <summary>
394 // Get the vehicle from the inventory based on the vehicle seatHeight
395 // </summary>
396 // <param name="vehicleSeatHeight">VehicleSeatHeight</param>
397 // <returns>The vehicle</returns>
398 public VehicleModel GetVehicleFromInventory(int vehicleSeatHeight)
399 {
400     // Loop through the inventory to find the correct vehicle
401     for (int i = 0; i < _inventory.Count; i++)
402     {
403         if (_inventory[i].SeatHeight == vehicleSeatHeight)
404         {
405             // Vehicle found
406             return _inventory[i];
407         }
408     }
409     // Vehicle not found
410     return null;
411 }
412
413 // <summary>
414 // Get the vehicle from the inventory based on the vehicle hasSidecar
415 // </summary>
416 // <param name="vehicleHasSidecar">VehicleHasSidecar</param>
417 // <returns>The vehicle</returns>
418 public VehicleModel GetVehicleFromInventory(bool vehicleHasSidecar)
419 {
420     // Loop through the inventory to find the correct vehicle
421     for (int i = 0; i < _inventory.Count; i++)
422     {
423         if (_inventory[i].HasSidecar == vehicleHasSidecar)
424         {
425             // Vehicle found
426             return _inventory[i];
427         }
428     }
429     // Vehicle not found
430     return null;
431 }
432
433 // <summary>
434 // Get the vehicle from the inventory based on the vehicle color
435 // </summary>
436 // <param name="vehicleColor">VehicleColor</param>
437 // <returns>The vehicle</returns>
438 public VehicleModel GetVehicleFromInventory(string vehicleColor)
439 {
440     // Loop through the inventory to find the correct vehicle
441     for (int i = 0; i < _inventory.Count; i++)
442     {
443         if (_inventory[i].Color == vehicleColor)
444         {
445             // Vehicle found
446             return _inventory[i];
447         }
448     }
449     // Vehicle not found
450     return null;
451 }
452
453 // <summary>
454 // Get the vehicle from the inventory based on the vehicle model
455 // </summary>
456 // <param name="vehicleModel">VehicleModel</param>
457 // <returns>The vehicle</returns>
458 public VehicleModel GetVehicleFromInventory(string vehicleModel)
459 {
460     // Loop through the inventory to find the correct vehicle
461     for (int i = 0; i < _inventory.Count; i++)
462     {
463         if (_inventory[i].Model == vehicleModel)
464         {
465             // Vehicle found
466             return _inventory[i];
467         }
468     }
469     // Vehicle not found
470     return null;
471 }
472
473 // <summary>
474 // Get the vehicle from the inventory based on the vehicle year
475 // </summary>
476 // <param name="vehicleYear">VehicleYear</param>
477 // <returns>The vehicle</returns>
478 public VehicleModel GetVehicleFromInventory(int vehicleYear)
479 {
480     // Loop through the inventory to find the correct vehicle
481     for (int i = 0; i < _inventory.Count; i++)
482     {
483         if (_inventory[i].Year == vehicleYear)
484         {
485             // Vehicle found
486             return _inventory[i];
487         }
488     }
489     // Vehicle not found
490     return null;
491 }
492
493 // <summary>
494 // Get the vehicle from the inventory based on the vehicle price
495 // </summary>
496 // <param name="vehiclePrice">VehiclePrice</param>
497 // <returns>The vehicle</returns>
498 public VehicleModel GetVehicleFromInventory(decimal vehiclePrice)
499 {
500     // Loop through the inventory to find the correct vehicle
501     for (int i = 0; i < _inventory.Count; i++)
502     {
503         if (_inventory[i].Price == vehiclePrice)
504         {
505             // Vehicle found
506             return _inventory[i];
507         }
508     }
509     // Vehicle not found
510     return null;
511 }
512
513 // <summary>
514 // Get the vehicle from the inventory based on the vehicle numWheels
515 // </summary>
516 // <param name="vehicleNumWheels">VehicleNumWheels</param>
517 // <returns>The vehicle</returns>
518 public VehicleModel GetVehicleFromInventory(int vehicleNumWheels)
519 {
520     // Loop through the inventory to find the correct vehicle
521     for (int i = 0; i < _inventory.Count; i++)
522     {
523         if (_inventory[i].NumWheels == vehicleNumWheels)
524         {
525             // Vehicle found
526             return _inventory[i];
527         }
528     }
529     // Vehicle not found
530     return null;
531 }
532
533 // <summary>
534 // Get the vehicle from the inventory based on the vehicle engineSize
535 // </summary>
536 // <param name="vehicleEngineSize">VehicleEngineSize</param>
537 // <returns>The vehicle</returns>
538 public VehicleModel GetVehicleFromInventory(string vehicleEngineSize)
539 {
540     // Loop through the inventory to find the correct vehicle
541     for (int i = 0; i < _inventory.Count; i++)
542     {
543         if (_inventory[i].EngineSize == vehicleEngineSize)
544         {
545             // Vehicle found
546             return _inventory[i];
547         }
548     }
549     // Vehicle not found
550     return null;
551 }
552
553 // <summary>
554 // Get the vehicle from the inventory based on the vehicle isConvertible
555 // </summary>
556 // <param name="vehicleIsConvertible">VehicleIsConvertible</param>
557 // <returns>The vehicle</returns>
558 public VehicleModel GetVehicleFromInventory(bool vehicleIsConvertible)
559 {
560     // Loop through the inventory to find the correct vehicle
561     for (int i = 0; i < _inventory.Count; i++)
562     {
563         if (_inventory[i].IsConvertible == vehicleIsConvertible)
564         {
565             // Vehicle found
566             return _inventory[i];
567         }
568     }
569     // Vehicle not found
570     return null;
571 }
572
573 // <summary>
574 // Get the vehicle from the inventory based on the vehicle trunkSize
575 // </summary>
576 // <param name="vehicleTrunkSize">VehicleTrunkSize</param>
577 // <returns>The vehicle</returns>
578 public VehicleModel GetVehicleFromInventory(int vehicleTrunkSize)
579 {
580     // Loop through the inventory to find the correct vehicle
581     for (int i = 0; i < _inventory.Count; i++)
582     {
583         if (_inventory[i].TrunkSize == vehicleTrunkSize)
584         {
585             // Vehicle found
586             return _inventory[i];
587         }
588     }
589     // Vehicle not found
590     return null;
591 }
592
593 // <summary>
594 // Get the vehicle from the inventory based on the vehicle hasBedCover
595 // </summary>
596 // <param name="vehicleHasBedCover">VehicleHasBedCover</param>
597 // <returns>The vehicle</returns>
598 public VehicleModel GetVehicleFromInventory(bool vehicleHasBedCover)
599 {
600     // Loop through the inventory to find the correct vehicle
601     for (int i = 0; i < _inventory.Count; i++)
602     {
603         if (_inventory[i].HasBedCover == vehicleHasBedCover)
604         {
605             // Vehicle found
606             return _inventory[i];
607         }
608     }
609     // Vehicle not found
610     return null;
611 }
612
613 // <summary>
614 // Get the vehicle from the inventory based on the vehicle bedSize
615 // </summary>
616 // <param name="vehicleBedSize">VehicleBedSize</param>
617 // <returns>The vehicle</returns>
618 public VehicleModel GetVehicleFromInventory(string vehicleBedSize)
619 {
620     // Loop through the inventory to find the correct vehicle
621     for (int i = 0; i < _inventory.Count; i++)
622     {
623         if (_inventory[i].BedSize == vehicleBedSize)
624         {
625             // Vehicle found
626             return _inventory[i];
627         }
628     }
629     // Vehicle not found
630     return null;
631 }
632
633 // <summary>
634 // Get the vehicle from the inventory based on the vehicle seatHeight
635 // </summary>
636 // <param name="vehicleSeatHeight">VehicleSeatHeight</param>
637 // <returns>The vehicle</returns>
638 public VehicleModel GetVehicleFromInventory(int vehicleSeatHeight)
639 {
640     // Loop through the inventory to find the correct vehicle
641     for (int i = 0; i < _inventory.Count; i++)
642     {
643         if (_inventory[i].SeatHeight == vehicleSeatHeight)
644         {
645             // Vehicle found
646             return _inventory[i];
647         }
648     }
649     // Vehicle not found
650     return null;
651 }
652
653 // <summary>
654 // Get the vehicle from the inventory based on the vehicle hasSidecar
655 // </summary>
656 // <param name="vehicleHasSidecar">VehicleHasSidecar</param>
657 // <returns>The vehicle</returns>
658 public VehicleModel GetVehicleFromInventory(bool vehicleHasSidecar)
659 {
660     // Loop through the inventory to find the correct vehicle
661     for (int i = 0; i < _inventory.Count; i++)
662     {
663         if (_inventory[i].HasSidecar == vehicleHasSidecar)
664         {
665             // Vehicle found
666             return _inventory[i];
667         }
668     }
669     // Vehicle not found
670     return null;
671 }
672
673 // <summary>
674 // Get the vehicle from the inventory based on the vehicle color
675 // </summary>
676 // <param name="vehicleColor">VehicleColor</param>
677 // <returns>The vehicle</returns>
678 public VehicleModel GetVehicleFromInventory(string vehicleColor)
679 {
680     // Loop through the inventory to find the correct vehicle
681     for (int i = 0; i < _inventory.Count; i++)
682     {
683         if (_inventory[i].Color == vehicleColor)
684         {
685             // Vehicle found
686             return _inventory[i];
687         }
688     }
689     // Vehicle not found
690     return null;
691 }
692
693 // <summary>
694 // Get the vehicle from the inventory based on the vehicle model
695 // </summary>
696 // <param name="vehicleModel">VehicleModel</param>
697 // <returns>The vehicle</returns>
698 public VehicleModel GetVehicleFromInventory(string vehicleModel)
699 {
700     // Loop through the inventory to find the correct vehicle
701     for (int i = 0; i < _inventory.Count; i++)
702     {
703         if (_inventory[i].Model == vehicleModel)
704         {
705             // Vehicle found
706             return _inventory[i];
707         }
708     }
709     // Vehicle not found
710     return null;
711 }
712
713 // <summary>
714 // Get the vehicle from the inventory based on the vehicle year
715 // </summary>
716 // <param name="vehicleYear">VehicleYear</param>
717 // <returns>The vehicle</returns>
718 public VehicleModel GetVehicleFromInventory(int vehicleYear)
719 {
720     // Loop through the inventory to find the correct vehicle
721     for (int i = 0; i < _inventory.Count; i++)
722     {
723         if (_inventory[i].Year == vehicleYear)
724         {
725             // Vehicle found
726             return _inventory[i];
727         }
728     }
729     // Vehicle not found
730     return null;
731 }
732
733 // <summary>
734 // Get the vehicle from the inventory based on the vehicle price
735 // </summary>
736 // <param name="vehiclePrice">VehiclePrice</param>
737 // <returns>The vehicle</returns>
738 public VehicleModel GetVehicleFromInventory(decimal vehiclePrice)
739 {
740     // Loop through the inventory to find the correct vehicle
741     for (int i = 0; i < _inventory.Count; i++)
742     {
743         if (_inventory[i].Price == vehiclePrice)
744         {
745             // Vehicle found
746             return _inventory[i];
747         }
748     }
749     // Vehicle not found
750     return null;
751 }
752
753 // <summary>
754 // Get the vehicle from the inventory based on the vehicle numWheels
755 // </summary>
756 // <param name="vehicleNumWheels">VehicleNumWheels</param>
757 // <returns>The vehicle</returns>
758 public VehicleModel GetVehicleFromInventory(int vehicleNumWheels)
759 {
760     // Loop through the inventory to find the correct vehicle
761     for (int i = 0; i < _inventory.Count; i++)
762     {
763         if (_inventory[i].NumWheels == vehicleNumWheels)
764         {
765             // Vehicle found
766             return _inventory[i];
767         }
768     }
769     // Vehicle not found
770     return null;
771 }
772
773 // <summary>
774 // Get the vehicle from the inventory based on the vehicle engineSize
775 // </summary>
776 // <param name="vehicleEngineSize">VehicleEngineSize</param>
777 // <returns>The vehicle</returns>
778 public VehicleModel GetVehicleFromInventory(string vehicleEngineSize)
779 {
780     // Loop through the inventory to find the correct vehicle
781     for (int i = 0; i < _inventory.Count; i++)
782     {
783         if (_inventory[i].EngineSize == vehicleEngineSize)
784         {
785             // Vehicle found
786             return _inventory[i];
787         }
788     }
789     // Vehicle not found
790     return null;
791 }
792
793 // <summary>
794 // Get the vehicle from the inventory based on the vehicle isConvertible
795 // </summary>
796 // <param name="vehicleIsConvertible">VehicleIsConvertible</param>
797 // <returns>The vehicle</returns>
798 public VehicleModel GetVehicleFromInventory(bool vehicleIsConvertible)
799 {
800     // Loop through the inventory to find the correct vehicle
801     for (int i = 0; i < _inventory.Count; i++)
802     {
803         if (_inventory[i].IsConvertible == vehicleIsConvertible)
804         {
805             // Vehicle found
806             return _inventory[i];
807         }
808     }
809     // Vehicle not found
810     return null;
811 }
812
813 // <summary>
814 // Get the vehicle from the inventory based on the vehicle trunkSize
815 // </summary>
816 // <param name="vehicleTrunkSize">VehicleTrunkSize</param>
817 // <returns>The vehicle</returns>
818 public VehicleModel GetVehicleFromInventory(int vehicleTrunkSize)
819 {
820     // Loop through the inventory to find the correct vehicle
821     for (int i = 0; i < _inventory.Count; i++)
822     {
823         if (_inventory[i].TrunkSize == vehicleTrunkSize)
824         {
825             // Vehicle found
826             return _inventory[i];
827         }
828     }
829     // Vehicle not found
830     return null;
831 }
832
833 // <summary>
834 // Get the vehicle from the inventory based on the vehicle hasBedCover
835 // </summary>
836 // <param name="vehicleHasBedCover">VehicleHasBedCover</param>
837 // <returns>The vehicle</returns>
838 public VehicleModel GetVehicleFromInventory(bool vehicleHasBedCover)
839 {
840     // Loop through the inventory to find the correct vehicle
841     for (int i = 0; i < _inventory.Count; i++)
842     {
843         if (_inventory[i].HasBedCover == vehicleHasBedCover)
844         {
845             // Vehicle found
846             return _inventory[i];
847         }
848     }
849     // Vehicle not found
850     return null;
851 }
852
853 // <summary>
854 // Get the vehicle from the inventory based on the vehicle bedSize
855 // </summary>
856 // <param name="vehicleBedSize">VehicleBedSize</param>
857 // <returns>The vehicle</returns>
858 public VehicleModel GetVehicleFromInventory(string vehicleBedSize)
859 {
860     // Loop through the inventory to find the correct vehicle
861     for (int i = 0; i < _inventory.Count; i++)
862     {
863         if (_inventory[i].BedSize == vehicleBedSize)
864         {
865             // Vehicle found
866             return _inventory[i];
867         }
868     }
869     // Vehicle not found
870     return null;
871 }
872
873 // <summary>
874 // Get the vehicle from the inventory based on the vehicle seatHeight
875 // </summary
```

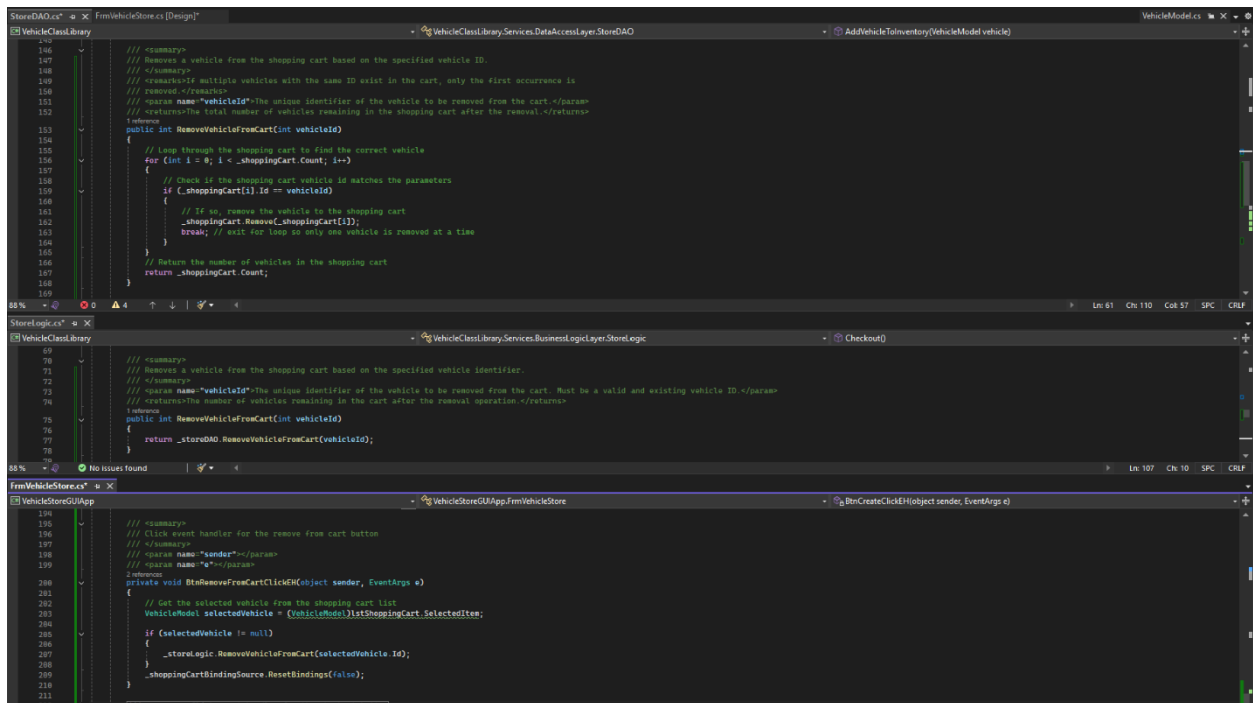


Figure 38: The three methods responsible for removing a vehicle from the cart in all three layers, the StoreDAO, the StoreLogic and the FrmVehicleStore

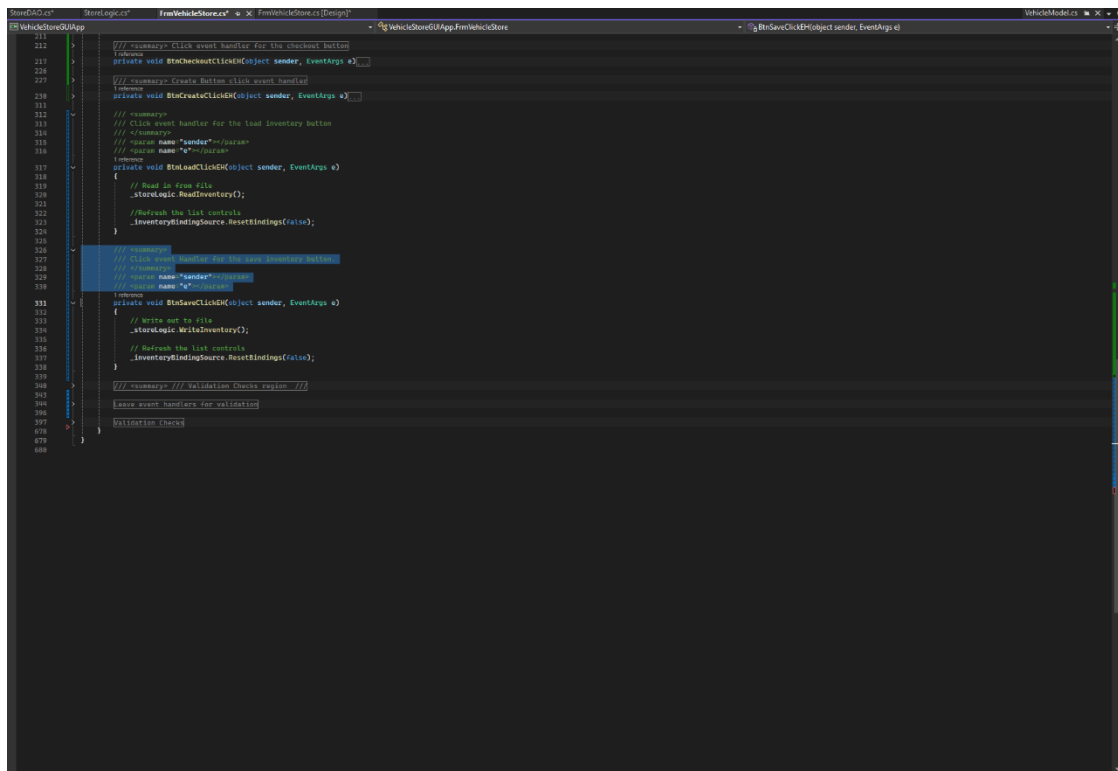


Figure 39: Implementation of the save and load functionality for the GUI

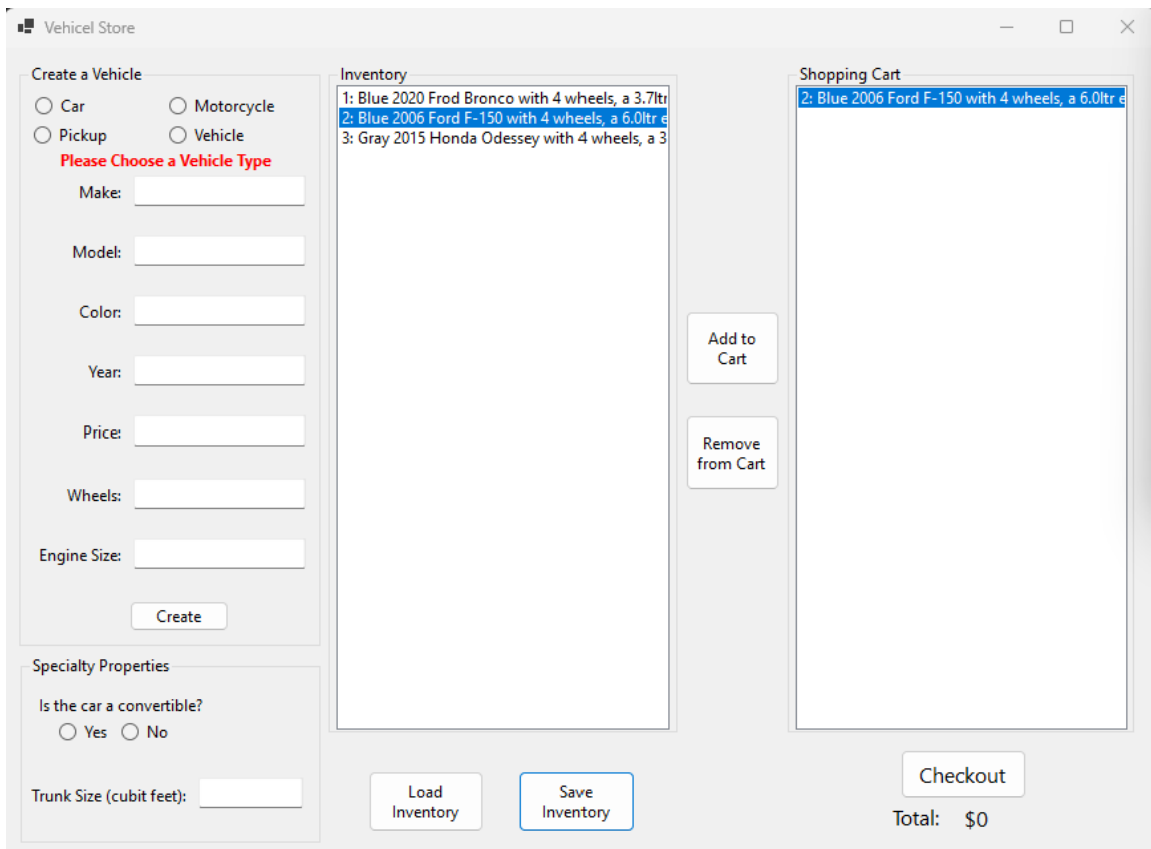


Figure 40: the GUI with all the new features implemented

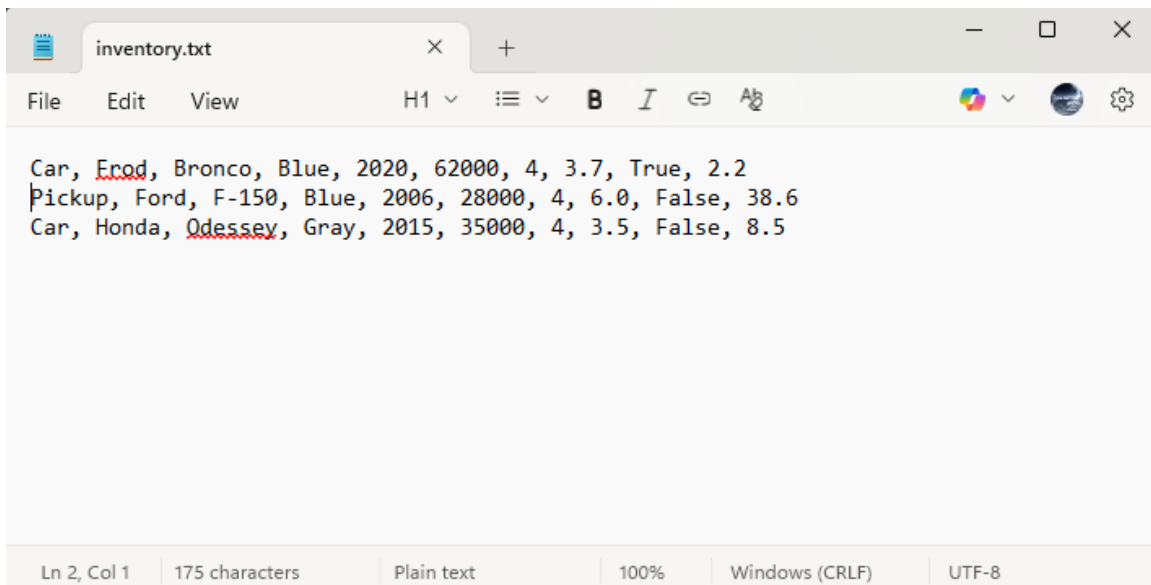


Figure 41: The contents of the save file after hitting save inventory in the above figure 40 screenshot

Part 3

Flowchart

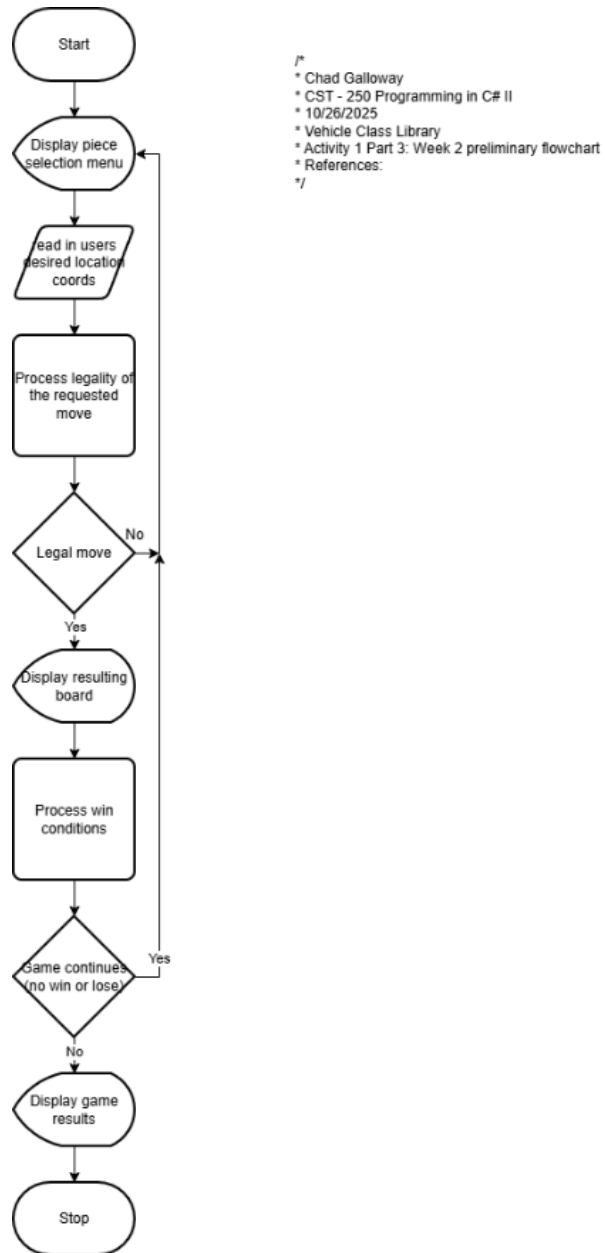


Figure 42: Screenshot Flowchart for week 2

UML

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

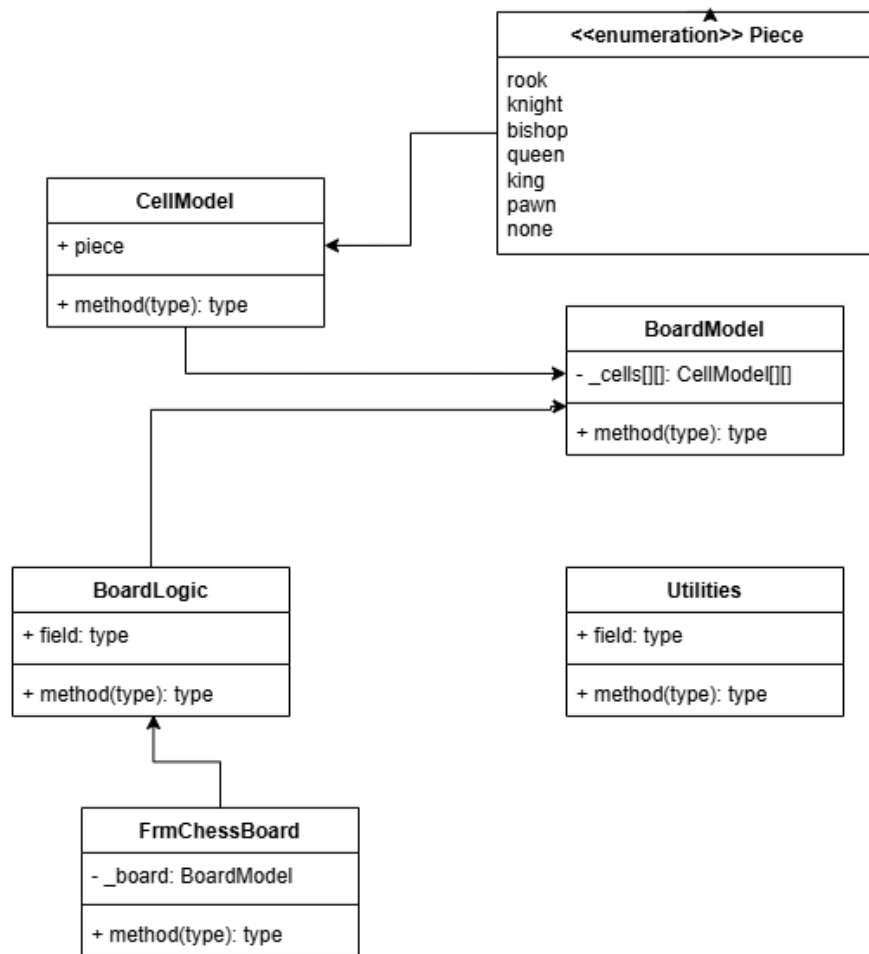


Figure 43: Screenshot of week 2 UML