

```
1 /*-----
2  * Author:      Dan Cassidy and Dr. Raman Adaikkalavan
3  * Date:        2015-06-17
4  * Assignment:  cView-P3
5  * Source File: ItemDBTest.cs
6  * Language:    C#
7  * Course:      CSCI-C 490, C# Programming, Mowe 08:00
8  * Project:     The overall goal of this project is to capitalize on the fact that government, from
9  *              local to national, has made some of its data open by developing a way to explore
10 *              this data and present it to a user in a meaningful fashion. This phase of the
11 *              project is meant to explore data from any combination of the Business dataset
12 *              (https://data.southbend.in.gov/d/imxu-7m5i), the Parks and Features dataset
13 *              (https://data.southbend.in.gov/d/yf5x-7tkb), and the Public Facility dataset
14 *              (https://data.southbend.in.gov/d/jeeef-dsq9).
15 * Purpose:     Small wrapper program for demonstrating the ItemDBInteractive class.
16 -----*/
17
18 using System;
19 using System.Collections.Generic;
20 using System.Linq;
21 using System.Text;
22 using System.Threading.Tasks;
23
24 namespace Ph3
25 {
26     public class ItemDBTest
27     {
28         /*-----
29          * Name:      Main
30          * Type:      Method
31          * Purpose:   Serves as the entry point to the program.
32          * Input:     (Ignored) string[] args, represents any command line arguments.
33          * Output:    Nothing.
34          -----*/
35         static void Main(string[] args)
36         {
37             // Declare a new ItemDBInteractive object and interactively manipulate said object.
38             ItemDBInteractive itemDBTest = new ItemDBInteractive();
39             itemDBTest.InteractiveManipulation();
40         }
41     }
42 }
43
```

```

1  /*-----
2  * Author:      Dan Cassidy
3  * Date:        2015-06-17
4  * Assignment:  cView-P3
5  * Source File: ItemDBInteractive.cs
6  * Language:    C#
7  * Course:      CSCI-C 490, C# Programming, MoWe 08:00
8  * Purpose:     Provides interactive management of an ItemDB object.
9  -----*/
10
11 using System;
12 using System.Collections.Generic;
13 using System.Globalization;
14 using System.IO;
15 using System.Linq;
16 using System.Text;
17 using System.Threading.Tasks;
18
19 namespace Ph3
20 {
21     public class ItemDBInteractive
22     {
23         /*-----
24         * Type:      Helper Constants
25         * Purpose:   Menu validation.
26         -----*/
27         private const MainMenu MainMenuMin = MainMenu.Load;
28         private const MainMenu MainMenuMax = MainMenu.Exit;
29         private const TypeMenu TypeMenuMin = TypeMenu.Business;
30         private const TypeMenu TypeMenuMax = TypeMenu.Back;
31
32         /*-----
33         * Name:      itemDB
34         * Type:      Field
35         * Purpose:   ItemDB that this class works with.
36         -----*/
37         private ItemDB itemDB = new ItemDB();
38
39         /*-----
40         * Name:      MainMenu
41         * Type:      Enum
42         * Purpose:   Enum for the main menu. Basic code idea from Stack Overflow.
43         *           http://stackoverflow.com/a/15752719
44         -----*/
45         private enum MainMenu
46         {
47             Load = 1,
48             Add,
49             Modify,
50             Search,
51             Delete,
52             DisplayAll,
53             Statistics,
54             Exit
55         }
56
57         /*-----
58         * Name:      TypeMenu
59         * Type:      Enum
60         * Purpose:   Enum for the type menu. Basic code idea from Stack Overflow.
61         *           http://stackoverflow.com/a/15752719
62         -----*/
63         private enum TypeMenu
64         {
65             Business = 1,
66             Park,

```

```

67         PublicFacility,
68         Back
69     }
70
71     /*-----
72     * Name:      InteractiveManipulation
73     * Type:      Method
74     * Purpose:   Entry point for interactive manipulation of ItemDB object.
75     * Input:     Nothing.
76     * Output:    Nothing.
77     -----*/
78     public void InteractiveManipulation()
79     {
80         //Loop the main menu until the user decides to exit.
81         while (MainMenuAction(MainMenuDisplay()) != MainMenu.Exit) ;
82     }
83
84     /*-----
85     * Name:      DataAdd
86     * Type:      Method
87     * Purpose:   Interactively add an item based on the user's input.
88     * Input:     Nothing.
89     * Output:    Nothing.
90     -----*/
91     private void DataAdd()
92     {
93         // Declare a reference for parent class.
94         Item itemToAdd;
95
96         // Prompt the user to choose what type of item to add.
97         Console.WriteLine("-----");
98         Console.WriteLine("| Add New Item |");
99         Console.WriteLine("-----");
100        TypeMenu choice = TypeMenuDisplay();
101
102        // Determine what the user wishes to do.
103        switch (choice)
104        {
105            case TypeMenu.Business:
106                itemToAdd = new Business();
107                Console.WriteLine("-----");
108                Console.WriteLine("| Add New Business |");
109                Console.WriteLine("-----");
110                break;
111
112            case TypeMenu.Park:
113                itemToAdd = new Park();
114                Console.WriteLine("-----");
115                Console.WriteLine("| Add New Park |");
116                Console.WriteLine("-----");
117                break;
118
119            case TypeMenu.PublicFacility:
120                itemToAdd = new PublicFacility();
121                Console.WriteLine("-----");
122                Console.WriteLine("| Add New Public Facility |");
123                Console.WriteLine("-----");
124                break;
125
126            case TypeMenu.Back:
127                // Nothing to do; user wants to go back.
128            default:
129                // Catch-all.
130                return;
131        }
132    }

```

```

133         // Handle filling in the common fields
134         Console.Write("Name: ");
135         itemToAdd.Name = Console.ReadLine();
136
137         Console.Write("Type: ");
138         itemToAdd.Type = Console.ReadLine();
139
140         Console.Write("Street Address: ");
141         itemToAdd.StreetAddress = Console.ReadLine();
142
143         Console.Write("City: ");
144         itemToAdd.City = Console.ReadLine();
145
146         Console.Write("State: ");
147         itemToAdd.State = Console.ReadLine();
148
149         Console.Write("ZIP Code: ");
150         itemToAdd.Zip = Console.ReadLine();
151
152         Console.Write("Latitude: ");
153         itemToAdd.Latitude = Console.ReadLine();
154
155         Console.Write("Longitude: ");
156         itemToAdd.Longitude = Console.ReadLine();
157
158         Console.Write("Phone Number: ");
159         itemToAdd.Phone = Console.ReadLine();
160
161         // Check whether the Item object is a Business object or Park object.
162         if (itemToAdd is Business)
163         {
164             // Business object. Handle Business object-specific fields.
165             Business businessToAdd = itemToAdd as Business;
166
167             Console.Write("Business License Fiscal Year: ");
168             businessToAdd.LicenseFiscalYear = SimpleConvert.ToInt32(Console.ReadLine());
169
170             Console.Write("Business License Number: ");
171             businessToAdd.LicenseNumber = SimpleConvert.ToInt32(Console.ReadLine());
172
173             Console.Write("Business License Issued Date: ");
174             businessToAdd.LicenseIssueDate = SimpleConvert.ToDateTime(Console.ReadLine());
175
176             Console.Write("Business License Expiration Date: ");
177             businessToAdd.LicenseExpirDate = SimpleConvert.ToDateTime(Console.ReadLine());
178
179             Console.Write("Business License Status: ");
180             businessToAdd.LicenseStatus = Console.ReadLine();
181
182             Console.Write("Council District: ");
183             businessToAdd.CouncilDistrict = Console.ReadLine();
184         }
185         else if (itemToAdd is Park)
186         {
187             // Park object. Handle Park object-specific fields.
188             Park parkToAdd = itemToAdd as Park;
189
190             Console.Write("# of Baseball Diamonds: ");
191             parkToAdd.FeatureBaseball = SimpleConvert.ToInt32(Console.ReadLine());
192
193             Console.Write("# of Basketball Courts: ");
194             parkToAdd.FeatureBasketball = SimpleConvert.ToSingle(Console.ReadLine());
195
196             Console.Write("# of Golf Courses: ");
197             parkToAdd.FeatureGolf = SimpleConvert.ToSingle(Console.ReadLine());
198

```

```

199         Console.Write("# of Large Multipurpose Fields: ");
200         parkToAdd.FeatureLargeMPField = SimpleConvert.ToInt32(Console.ReadLine());
201
202         Console.Write("# of Tennis Courts: ");
203         parkToAdd.FeatureTennis = SimpleConvert.ToInt32(Console.ReadLine());
204
205         Console.Write("# of Volleyball Courts: ");
206         parkToAdd.FeatureVolleyball = SimpleConvert.ToInt32(Console.ReadLine());
207     }
208
209     // Extra line for formatting.
210     Console.WriteLine();
211
212     // Add the new item to the main data set.
213     itemDB.Add(itemToAdd);
214 }
215
216 /*-----
217  * Name:      DataDelete
218  * Type:      Method
219  * Purpose:   Interactively deletes an object based upon user input.
220  * Input:     Nothing.
221  * Output:    Nothing.
222 -----*/
223 private void DataDelete()
224 {
225     // Display the user's choice.
226     Console.WriteLine("-----");
227     Console.WriteLine("| Delete Item -- Existing Items |");
228     Console.WriteLine("-----");
229
230     // Display a simple list of all the objects in the data set.
231     itemDB.DisplayAll(true);
232
233     // Get the user's choice of which object to delete.
234     Console.Write("Select item ID (0 to cancel): ");
235     int itemIDToDelete = SimpleConvert.ToInt32(Console.ReadLine());
236     int indexToDelete = itemDB.GetItemIndex(itemIDToDelete);
237
238     // Extra line for formatting.
239     Console.WriteLine();
240
241     // Display the results.
242     Console.WriteLine("-----");
243     Console.WriteLine("| Delete Item -- Results |");
244     Console.WriteLine("-----");
245
246     // Validate the user's choice.
247     if (itemIDToDelete == 0)
248     {
249         // The user changed their mind.
250         Console.WriteLine("Cancelled.\n");
251         return;
252     }
253     else if (itemIDToDelete < 0 || indexToDelete < 0)
254     {
255         // The user input an invalid object index.
256         Console.WriteLine("Invalid item.\n");
257         return;
258     }
259
260     // Delete the object and display confirmation of its deletion.
261     if (itemDB.Delete(itemIDToDelete))
262         Console.WriteLine("Item ID {0} has been deleted.\n", itemIDToDelete);
263     else
264         Console.WriteLine("Error occured while attempting to delete item ID {0}.\n",

```

```

265         itemIDToDelete);
266
267         // Display a simple list of the still existing items.
268         itemDB.DisplayAll(true);
269     }
270
271     /*-----
272     * Name:      DataDisplayAll
273     * Type:      Method
274     * Purpose: Displays a list of all items.
275     * Input:     Nothing.
276     * Output:    Nothing.
277     -----*/
278     private void DataDisplayAll()
279     {
280         // Display the user's choice.
281         Console.WriteLine("-----");
282         Console.WriteLine("| Display All Items |");
283         Console.WriteLine("-----");
284
285         // Display all the items.
286         itemDB.DisplayAll();
287     }
288
289     /*-----
290     * Name:      DataLoad
291     * Type:      Method
292     * Purpose: Get the user's choice of CSV files to import.
293     * Input:     Nothing.
294     * Output:    Nothing.
295     -----*/
296     private void DataLoad()
297     {
298         TypeMenu typeChoice;
299
300         itemDB.Reset();
301
302         // Display the user's choice.
303         Console.WriteLine("-----");
304         Console.WriteLine("| Load Files |");
305         Console.WriteLine("-----");
306
307         // Read files for as long as the user wants.
308         while ((typeChoice = TypeMenuDisplay()) != TypeMenu.Back)
309         {
310             Console.Write("Enter a filename to load: ");
311
312             try
313             {
314                 int tempCount = itemDB.Count;
315                 DataLoadProcessFile(Console.ReadLine(), typeChoice);
316                 Console.WriteLine("{0} item{1} loaded.", itemDB.Count - tempCount,
317                     (itemDB.Count - tempCount != 1) ? "s" : "");
318             }
319             catch (Exception ex)
320             {
321                 Console.WriteLine(ex.Message);
322             }
323
324             Console.WriteLine();
325         }
326     }
327
328     /*-----
329     * Name:      DataLoadProcessFile
330     * Type:      Method

```

```

331     * Purpose: Process the file specified and add the resulting Items to the ItemDB. Used some
332     *           of the code from the book example Fig17_11 as a starting point.
333     * Input:   string fileName, contains the filename to process.
334     * Input:   TypeMenu itemType, contains the type of item to add.
335     * Output:  Nothing.
336     -----*/
337 private void DataLoadProcessFile(string fileName, TypeMenu itemType)
338 {
339     using (StreamReader fileReader = new StreamReader(fileName))
340     {
341         string inputItem = fileReader.ReadLine();
342         string[] inputFields;
343
344         while (inputItem != null)
345         {
346             Item toAdd = null;
347             inputFields = inputItem.Split(',');
348
349             // Process a line based on what type is being imported.
350             switch (itemType)
351             {
352                 case TypeMenu.Business:
353                     toAdd = new Business(inputFields[0], inputFields[1], inputFields[2],
354                     inputFields[3], inputFields[4], inputFields[5], inputFields[6],
355                     inputFields[7], inputFields[8],
356                     SimpleConvert.ToInt32(inputFields[9].Split('-')[0]),
357                     SimpleConvert.ToInt32(inputFields[9].Split('-')[1]),
358                     SimpleConvert.ToDateTime(inputFields[10]),
359                     SimpleConvert.ToDateTime(inputFields[11]), inputFields[12],
360                     inputFields[13]);
361                     break;
362
363                 case TypeMenu.Park:
364                     toAdd = new Park(inputFields[0], inputFields[1], inputFields[2],
365                     inputFields[3], inputFields[4], inputFields[5], inputFields[6],
366                     inputFields[7], inputFields[8],
367                     SimpleConvert.ToInt32(inputFields[9]),
368                     SimpleConvert.ToSingle(inputFields[10]),
369                     SimpleConvert.ToSingle(inputFields[11]),
370                     SimpleConvert.ToInt32(inputFields[12]),
371                     SimpleConvert.ToInt32(inputFields[13]),
372                     SimpleConvert.ToInt32(inputFields[14]));
373                     break;
374
375                 case TypeMenu.PublicFacility:
376                     toAdd = new PublicFacility(inputFields[0], inputFields[1],
377                     inputFields[2], inputFields[3], inputFields[4], inputFields[5],
378                     inputFields[6], inputFields[7], inputFields[8]);
379                     break;
380
381                 default:
382                     break;
383             }
384
385             if (toAdd != null)
386                 itemDB.Add(toAdd);
387
388             inputItem = fileReader.ReadLine();
389         }
390     }
391 }
392
393 /*-----
394  * Name:      DataModify
395  * Type:      Method
396  * Purpose:   Interactively modifies an object based on the user's input.

```

```

397     * Input:  Nothing.
398     * Output: Nothing.
399     -----*/
400 private void DataModify()
401 {
402     //Display the user's choice.
403     Console.WriteLine("-----");
404     Console.WriteLine("| Modify Item -- Existing Items |");
405     Console.WriteLine("-----");
406
407     // Display a simple list of all the objects in the data set.
408     itemDB.DisplayAll(true);
409
410     // Get the user's choice of which object to delete.
411     Console.Write("Select item ID (0 to cancel): ");
412     int itemIDToModify = SimpleConvert.ToInt32(Console.ReadLine());
413     int indexToModify = itemDB.GetItemIndex(itemIDToModify);
414
415     // Extra line for formatting.
416     Console.WriteLine();
417
418     // Validate the user's choice.
419     if (itemIDToModify == 0)
420     {
421         // The user changed their mind.
422         Console.WriteLine("Cancelled.\n");
423         return;
424     }
425     else if (itemIDToModify < 0 || indexToModify < 0)
426     {
427         // The user input an invalid object index.
428         Console.WriteLine("Invalid item.\n");
429         return;
430     }
431
432     // Store reference to item copy.
433     Item itemToModify = itemDB.GetItem(itemIDToModify);
434
435     do
436     {
437         // Display the chosen object.
438         Console.WriteLine("-----");
439         Console.WriteLine("| Modify Item -- Chosen Item |");
440         Console.WriteLine("-----");
441         Console.WriteLine("{0}\n", itemToModify);
442
443         // Loop while the use has not chosen to go back.
444     } while (DataModifyMenuAction(FieldMenuDisplay(itemToModify),
445         itemToModify) != Item.FieldMenuHelper.Back);
446 }
447
448 /*-----
449  * Name:    DataModifyMenuAction
450  * Type:    Method
451  * Purpose: Acts on the user's choice made at the Modify Menu.
452  * Input:   Item.FieldMenuHelper choice, represents the action specified.
453  * Input:   Item itemToModify, is a copy of the object that will be modified.
454  * Output:  Item.FieldMenuHelper, represents the action specified.
455  -----*/
456 private Item.FieldMenuHelper DataModifyMenuAction(Item.FieldMenuHelper choice,
457     Item itemToModify)
458 {
459     // Handle the common fields of an Item object.
460     switch (choice)
461     {
462         case Item.FieldMenuHelper.Name:

```



```

463         // Change the Name of the item.
464         Console.WriteLine("Current Name: {0}", itemToModify.Name);
465         Console.Write("New Name: ");
466         itemToModify.Name = Console.ReadLine();
467         break;
468
469     case Item.FieldMenuHelper.Type:
470         // Change the Type of the item.
471         Console.WriteLine("Current Type: {0}", itemToModify.Type);
472         Console.Write("New Type: ");
473         itemToModify.Type = Console.ReadLine();
474         break;
475
476     case Item.FieldMenuHelper.StreetAddress:
477         // Change the StreetAddress of the item.
478         Console.WriteLine("Current Street Address: {0}", itemToModify.StreetAddress);
479         Console.Write("New Street Address: ");
480         itemToModify.StreetAddress = Console.ReadLine();
481         break;
482
483     case Item.FieldMenuHelper.City:
484         // Change the City of the item.
485         Console.WriteLine("Current City: {0}", itemToModify.City);
486         Console.Write("New City: ");
487         itemToModify.City = Console.ReadLine();
488         break;
489
490     case Item.FieldMenuHelper.State:
491         // Change the State of the item.
492         Console.WriteLine("Current State: {0}", itemToModify.State);
493         Console.Write("New State: ");
494         itemToModify.State = Console.ReadLine();
495         break;
496
497     case Item.FieldMenuHelper.Zip:
498         // Change the Zip of the item.
499         Console.WriteLine("Current ZIP Code: {0}", itemToModify.Zip);
500         Console.Write("New ZIP Code: ");
501         itemToModify.Zip = Console.ReadLine();
502         break;
503
504     case Item.FieldMenuHelper.Latitude:
505         // Change the Latitude of the item.
506         Console.WriteLine("Current Latitude: {0}", itemToModify.Latitude);
507         Console.Write("New Latitude: ");
508         itemToModify.Latitude = Console.ReadLine();
509         break;
510
511     case Item.FieldMenuHelper.Longitude:
512         // Change the Longitude of the item.
513         Console.WriteLine("Current Longitude: {0}", itemToModify.Longitude);
514         Console.Write("New Longitude: ");
515         itemToModify.Longitude = Console.ReadLine();
516         break;
517
518     case Item.FieldMenuHelper.Phone:
519         // Change the Phone of the item.
520         Console.WriteLine("Current Phone Number: {0}", itemToModify.Phone);
521         Console.Write("New Phone Number: ");
522         itemToModify.Phone = Console.ReadLine();
523         break;
524
525     case Item.FieldMenuHelper.Back:
526     case Item.FieldMenuHelper.BackBusiness:
527     case Item.FieldMenuHelper.BackPark:
528         // Nothing to do; the user wants to go back.

```

```

529         return Item.FieldMenuHelper.Back;
530
531     default:
532         // Catch-all.
533         break;
534 }
535
536 // Check whether the Item object is a Business object or Park object.
537 if (itemToModify is Business)
538 {
539     // Business object. Handle Business object-specific fields.
540     Business businessToModify = itemToModify as Business;
541
542     switch (choice)
543     {
544         case Item.FieldMenuHelper.LicenseFiscalYear:
545             // Change the LicenseFiscalYear of the business.
546             Console.WriteLine("Current Business License Fiscal Year: {0}",
547                 businessToModify.LicenseFiscalYear);
548             Console.Write("New Business License Fiscal Year: ");
549             businessToModify.LicenseFiscalYear =
550                 SimpleConvert.ToInt32(Console.ReadLine());
551             break;
552
553         case Item.FieldMenuHelper.LicenseNumber:
554             // Change the LicenseNumber of the business.
555             Console.WriteLine("Current Business License Number: {0}",
556                 businessToModify.LicenseNumber);
557             Console.Write("New Business License Number: ");
558             businessToModify.LicenseNumber = SimpleConvert.ToInt32(Console.ReadLine());
559             break;
560
561         case Item.FieldMenuHelper.LicenseIssueDate:
562             // Change the LicenseIssueDate of the business.
563             Console.WriteLine("Current Business License Issue Date: {0}",
564                 businessToModify.LicenseIssueDate.ToShortDateString());
565             Console.Write("New Business License Issue Date: ");
566             businessToModify.LicenseIssueDate =
567                 SimpleConvert.ToDateTime(Console.ReadLine());
568             break;
569
570         case Item.FieldMenuHelper.LicenseExpirDate:
571             // Change the LicenseExpirDate of the business.
572             Console.WriteLine("Current Business License Expiration Date: {0}",
573                 businessToModify.LicenseExpirDate.ToShortDateString());
574             Console.Write("New Business License Expiration Date: ");
575             businessToModify.LicenseExpirDate =
576                 SimpleConvert.ToDateTime(Console.ReadLine());
577             break;
578
579         case Item.FieldMenuHelper.LicenseStatus:
580             // Change the LicenseStatus of the business.
581             Console.WriteLine("Current Business License Status: {0}",
582                 businessToModify.LicenseStatus);
583             Console.Write("New Business License Status: ");
584             businessToModify.LicenseStatus = Console.ReadLine();
585             break;
586
587         case Item.FieldMenuHelper.CouncilDistrict:
588             // Change the CouncilDistrict of the business.
589             Console.WriteLine("Current Council District: {0}",
590                 businessToModify.CouncilDistrict);
591             Console.Write("New Council District: ");
592             businessToModify.CouncilDistrict = Console.ReadLine();
593             break;
594

```

```
595         default:
596             // Catch-all.
597             break;
598     }
599 }
600 else if (itemToModify is Park)
601 {
602     // Park object. Handle Park object-specific fields.
603     Park parkToModify = itemToModify as Park;
604
605     switch (choice)
606     {
607         case Item.FieldMenuHelper.FeatureBaseball:
608             // Change the FeatureBaseball of the park.
609             Console.WriteLine("Current # of Baseball Diamonds: {0}",
610                 parkToModify.FeatureBaseball);
611             Console.Write("New # of Baseball Diamonds: ");
612             parkToModify.FeatureBaseball = SimpleConvert.ToInt32(Console.ReadLine());
613             break;
614
615         case Item.FieldMenuHelper.FeatureBasketball:
616             // Change the FeatureBasketball of the park.
617             Console.WriteLine("Current # of Basketball Courts: {0}",
618                 parkToModify.FeatureBasketball);
619             Console.Write("New # of Basketball Courts: ");
620             parkToModify.FeatureBasketball = SimpleConvert.ToSingle(Console.ReadLine());
621             break;
622
623         case Item.FieldMenuHelper.FeatureGolf:
624             // Change the Type of the park.
625             Console.WriteLine("Current # of Golf Courses: {0}",
626                 parkToModify.FeatureGolf);
627             Console.Write("New # of Golf Courses: ");
628             parkToModify.FeatureGolf = SimpleConvert.ToSingle(Console.ReadLine());
629             break;
630
631         case Item.FieldMenuHelper.FeatureLargeMPField:
632             // Change the Type of the park.
633             Console.WriteLine("Current # of Large Multipurpose Fields: {0}",
634                 parkToModify.FeatureLargeMPField);
635             Console.Write("New # of Large Multipurpose Fields: ");
636             parkToModify.FeatureLargeMPField =
637                 SimpleConvert.ToInt32(Console.ReadLine());
638             break;
639
640         case Item.FieldMenuHelper.FeatureTennis:
641             // Change the Type of the park.
642             Console.WriteLine("Current # of Tennis Courts: {0}",
643                 parkToModify.FeatureTennis);
644             Console.Write("New # of Tennis Courts: ");
645             parkToModify.FeatureTennis = SimpleConvert.ToInt32(Console.ReadLine());
646             break;
647
648         case Item.FieldMenuHelper.FeatureVolleyball:
649             // Change the Type of the park.
650             Console.WriteLine("Current # of Volleyball Courts: {0}",
651                 parkToModify.FeatureVolleyball);
652             Console.Write("New # of Volleyball Courts: ");
653             parkToModify.FeatureVolleyball = SimpleConvert.ToInt32(Console.ReadLine());
654             break;
655
656         default:
657             // Catch-all.
658             break;
659     }
660 }
```

```

661
662         // Modify the item in itemDB.
663         itemDB.Modify(itemToModify);
664
665         // Extra line for formatting.
666         Console.WriteLine();
667
668         // Return choice so the calling method knows what the choice was and can act
669         // accordingly.
670         return choice;
671     }
672
673     /*-----
674     * Name:    DataSave
675     * Type:    Method
676     * Purpose: Save the data in itemDB before exiting.
677     * Input:   Nothing.
678     * Output:  Nothing.
679     -----*/
680     private void DataSave()
681     {
682         // Display user's choice.
683         Console.WriteLine("-----");
684         Console.WriteLine("| Save and Exit |");
685         Console.WriteLine("-----");
686
687         if (itemDB.IsChanged)
688         {
689             // ItemDB has been changed, ask the user if they wish to save and get response.
690             bool validInput;
691             Console.Write("Changes detected in the item list, do you wish to save? [Y]/N");
692             do
693             {
694                 validInput = false;
695                 ConsoleKeyInfo keyPress = Console.ReadKey(true);
696                 switch (keyPress.Key)
697                 {
698                     case ConsoleKey.Enter:
699                         if (keyPress.Modifiers == 0)
700                             // User pressed Enter; continue with save.
701                             validInput = true;
702                         break;
703
704                     case ConsoleKey.Y:
705                         if (keyPress.Modifiers == 0 ||
706                             keyPress.Modifiers == ConsoleModifiers.Shift)
707                             // User pressed 'Y'; continue with save.
708                             validInput = true;
709                         break;
710
711                     case ConsoleKey.N:
712                         if (keyPress.Modifiers == 0 ||
713                             keyPress.Modifiers == ConsoleModifiers.Shift)
714                         {
715                             // User pressed 'N'; abort save.
716                             Console.WriteLine();
717                             return;
718                         }
719                         break;
720
721                     default:
722                         break;
723                 }
724                 // Loop while invalid input.
725             } while (!validInput);
726

```

```
727         Console.WriteLine("\n\n!!!WARNING!!! Any file you choose will be OVERWRITTEN.");
728
729         string fileNameBusinesses = "";
730         string fileNameParks = "";
731         string fileNamePublicFacilities = "";
732         bool saveSuccess = false;
733
734         // Utilize the search function to create item DBs of each type of item.
735         ItemDB allBusinesses = itemDB.Search(
736             "",
737             Enum.GetName(typeof(TypeMenu), TypeMenu.Business).ToLower(),
738             Item.FieldMenuHelper.Name);
739         ItemDB allParks = itemDB.Search(
740             "",
741             Enum.GetName(typeof(TypeMenu), TypeMenu.Park).ToLower(),
742             Item.FieldMenuHelper.Name);
743         ItemDB allPublicFacilities = itemDB.Search(
744             "",
745             Enum.GetName(typeof(TypeMenu), TypeMenu.PublicFacility).ToLower(),
746             Item.FieldMenuHelper.Name);
747
748         // If the DBs aren't empty, ask for a filename for that item type.
749         if (allBusinesses.Count != 0)
750         {
751             Console.Write("Please choose a filename for business items: ");
752             fileNameBusinesses = Console.ReadLine();
753         }
754         if (allParks.Count != 0)
755         {
756             Console.Write("Please choose a filename for park items: ");
757             fileNameParks = Console.ReadLine();
758         }
759         if (allPublicFacilities.Count != 0)
760         {
761             Console.Write("Please choose a filename for public facility items: ");
762             fileNamePublicFacilities = Console.ReadLine();
763         }
764
765         Console.WriteLine();
766
767         // Attempt to save the business data.
768         try
769         {
770             if (fileNameBusinesses != "")
771                 using (StreamWriter fileWriter = new StreamWriter(fileNameBusinesses))
772                 {
773                     foreach (var item in allBusinesses)
774                         fileWriter.WriteLine(item.ToStringCSV());
775                     Console.WriteLine("Business data saved successfully.");
776                     saveSuccess = true;
777                 }
778         }
779         catch (Exception ex)
780         {
781             Console.WriteLine("Error attempting to save business data:");
782             Console.WriteLine(ex.Message);
783         }
784
785         // Attempt to save the park data.
786         try
787         {
788             if (fileNameParks != "")
789                 using (StreamWriter fileWriter = new StreamWriter(fileNameParks))
790                 {
791                     foreach (var item in allParks)
792                         fileWriter.WriteLine(item.ToStringCSV());
```

```

793         Console.WriteLine("Park data saved successfully.");
794         saveSuccess = true;
795     }
796 }
797 catch (Exception ex)
798 {
799     Console.WriteLine("Error attempting to save park data:");
800     Console.WriteLine(ex.Message);
801 }
802
803 // Attempt to save the public facility data.
804 try
805 {
806     if (fileNamePublicFacilities != "")
807         using (StreamWriter fileWriter = new StreamWriter(fileNamePublicFacilities))
808         {
809             foreach (var item in allPublicFacilities)
810                 fileWriter.WriteLine(item.ToStringCSV());
811             Console.WriteLine("Public facility data saved successfully.");
812             saveSuccess = true;
813         }
814     }
815 catch (Exception ex)
816 {
817     Console.WriteLine("Error attempting to save public facility data:");
818     Console.WriteLine(ex.Message);
819 }
820
821 if (saveSuccess)
822     Console.WriteLine();
823 }
824 else
825 {
826     // ItemDB has not been changed.
827     Console.WriteLine("No changes to save.\n");
828 }
829 }
830
831 /*-----
832  * Name:    DataSearch
833  * Type:    Method
834  * Purpose: Interactively searches for objects based upon user input.
835  * Input:   Nothing.
836  * Output:  Nothing.
837  -----*/
838 private void DataSearch()
839 {
840     TypeMenu typeChoice;
841
842     do
843     {
844         // Display the user's choice.
845         Console.WriteLine("-----");
846         Console.WriteLine("| Search Items |");
847         Console.WriteLine("-----");
848         typeChoice = TypeMenuDisplay();
849
850         if (typeChoice != TypeMenu.Back)
851         {
852             do
853             {
854                 // Display the user's choice.
855                 switch (typeChoice)
856                 {
857                     case TypeMenu.Business:
858                         Console.WriteLine("-----");

```

```

859         Console.WriteLine("| Search Businesses |");
860         Console.WriteLine("-----");
861         break;
862
863         case TypeMenu.Park:
864             Console.WriteLine("-----");
865             Console.WriteLine("| Search Parks |");
866             Console.WriteLine("-----");
867             break;
868
869         case TypeMenu.PublicFacility:
870             Console.WriteLine("-----");
871             Console.WriteLine("| Search Public Facilities |");
872             Console.WriteLine("-----");
873             break;
874
875         case TypeMenu.Back:
876             // Nothing to do; user wants to go back.
877         default:
878             // Catch-all.
879             break;
880     }
881
882     // Loop while the user has not chosen to go back.
883     } while (DataSearchMenuAction(FieldMenuDisplay(typeChoice), typeChoice) !=
884             Item.FieldMenuHelper.Back);
885 }
886 // Loop while the user has not chosen to go back.
887 } while (typeChoice != TypeMenu.Back);
888 }
889
890 /*-----
891  * Name:    DataSearchMenuAction
892  * Type:    Method
893  * Purpose: Acts on the user's choice made at the Search Menu.
894  * Input:   Item.FieldMenuHelper field, represents the action specified.
895  * Input:   TypeMenu type, represents the type of item the user is searching for.
896  * Output:  Item.FieldMenuHelper, represents the action specified.
897  -----*/
898 private Item.FieldMenuHelper DataSearchMenuAction(Item.FieldMenuHelper field, TypeMenu type)
899 {
900     // Decide what to display based on the user's type.
901     switch (field)
902     {
903         case Item.FieldMenuHelper.Name:
904             // Search the Name property.
905             Console.WriteLine("-----");
906             Console.WriteLine("| Search Items -- Name |");
907             Console.WriteLine("-----");
908             break;
909
910         case Item.FieldMenuHelper.Type:
911             // Search the Type property.
912             Console.WriteLine("-----");
913             Console.WriteLine("| Search Items -- Type |");
914             Console.WriteLine("-----");
915             break;
916
917         case Item.FieldMenuHelper.StreetAddress:
918             // Search the StreetAddress property.
919             Console.WriteLine("-----");
920             Console.WriteLine("| Search Items -- Street Address |");
921             Console.WriteLine("-----");
922             break;
923
924         case Item.FieldMenuHelper.City:

```

```
925         // Search the City property.
926         Console.WriteLine("-----");
927         Console.WriteLine("| Search Items -- City |");
928         Console.WriteLine("-----");
929         break;
930
931     case Item.FieldMenuHelper.State:
932         // Search the State property.
933         Console.WriteLine("-----");
934         Console.WriteLine("| Search Items -- State |");
935         Console.WriteLine("-----");
936         break;
937
938     case Item.FieldMenuHelper.Zip:
939         // Search the Zip property.
940         Console.WriteLine("-----");
941         Console.WriteLine("| Search Items -- ZIP Code |");
942         Console.WriteLine("-----");
943         break;
944
945     case Item.FieldMenuHelper.Latitude:
946         // Search the Latitude property.
947         Console.WriteLine("-----");
948         Console.WriteLine("| Search Items -- Latitude |");
949         Console.WriteLine("-----");
950         break;
951
952     case Item.FieldMenuHelper.Longitude:
953         // Search the Longitude property.
954         Console.WriteLine("-----");
955         Console.WriteLine("| Search Items -- Longitude |");
956         Console.WriteLine("-----");
957         break;
958
959     case Item.FieldMenuHelper.Phone:
960         // Search the Phone property.
961         Console.WriteLine("-----");
962         Console.WriteLine("| Search Items -- Phone Number |");
963         Console.WriteLine("-----");
964         break;
965
966     case Item.FieldMenuHelper.LicenseFiscalYear:
967         // Search the LicenseFiscalYear property.
968         Console.WriteLine("-----");
969         Console.WriteLine("| Search Items -- Business License Fiscal Year |");
970         Console.WriteLine("-----");
971         break;
972
973     case Item.FieldMenuHelper.LicenseNumber:
974         // Search the LicenseNumber property.
975         Console.WriteLine("-----");
976         Console.WriteLine("| Search Items -- Business License Number |");
977         Console.WriteLine("-----");
978         break;
979
980     case Item.FieldMenuHelper.LicenseIssueDate:
981         // Search the LicenseIssueDate property.
982         Console.WriteLine("-----");
983         Console.WriteLine("| Search Items -- Business License Issue Date |");
984         Console.WriteLine("-----");
985         break;
986
987     case Item.FieldMenuHelper.LicenseExpirDate:
988         // Search the LicenseExpirDate property.
989         Console.WriteLine("-----");
990         Console.WriteLine("| Search Items -- Business License Expiration Date |");
```



```
991         Console.WriteLine("-----");
992         break;
993
994     case Item.FieldMenuHelper.LicenseStatus:
995         // Search the LicenseStatus property.
996         Console.WriteLine("-----");
997         Console.WriteLine("| Search Items -- Business License Status |");
998         Console.WriteLine("-----");
999         break;
1000
1001     case Item.FieldMenuHelper.CouncilDistrict:
1002         // Search the CouncilDistrict property.
1003         Console.WriteLine("-----");
1004         Console.WriteLine("| Search Items -- Council District |");
1005         Console.WriteLine("-----");
1006         break;
1007
1008     case Item.FieldMenuHelper.FeatureBaseball:
1009         // Search the FeatureBaseball property.
1010         Console.WriteLine("-----");
1011         Console.WriteLine("| Search Items -- # of Baseball Diamonds |");
1012         Console.WriteLine("-----");
1013         break;
1014
1015     case Item.FieldMenuHelper.FeatureBasketball:
1016         // Search the FeatureBasketball property.
1017         Console.WriteLine("-----");
1018         Console.WriteLine("| Search Items -- # of Basketball Courts |");
1019         Console.WriteLine("-----");
1020         break;
1021
1022     case Item.FieldMenuHelper.FeatureGolf:
1023         // Search the FeatureGolf property.
1024         Console.WriteLine("-----");
1025         Console.WriteLine("| Search Items -- # of Golf Courses |");
1026         Console.WriteLine("-----");
1027         break;
1028
1029     case Item.FieldMenuHelper.FeatureLargeMPField:
1030         // Search the FeatureLargeMPField property.
1031         Console.WriteLine("-----");
1032         Console.WriteLine("| Search Items -- # of Large Multipurpose Fields |");
1033         Console.WriteLine("-----");
1034         break;
1035
1036     case Item.FieldMenuHelper.FeatureTennis:
1037         // Search the FeatureTennis property.
1038         Console.WriteLine("-----");
1039         Console.WriteLine("| Search Items -- # of Tennis Courts |");
1040         Console.WriteLine("-----");
1041         break;
1042
1043     case Item.FieldMenuHelper.FeatureVolleyball:
1044         // Search the FeatureVolleyball property.
1045         Console.WriteLine("-----");
1046         Console.WriteLine("| Search Items -- # of Volleyball Courts |");
1047         Console.WriteLine("-----");
1048         break;
1049
1050     case Item.FieldMenuHelper.Back:
1051     case Item.FieldMenuHelper.BackBusiness:
1052     case Item.FieldMenuHelper.BackPark:
1053         // Nothing to do; the user wants to go back.
1054     default:
1055         // Catch-all.
1056         return Item.FieldMenuHelper.Back;
```

```

1057     }
1058
1059     Console.WriteLine("What kind of comparator do you wish to use?");
1060     Console.WriteLine(" | - contains (default)    >= - greater than or equal to");
1061     Console.WriteLine(" !| - does not contain      <= - less than or equal to");
1062     Console.WriteLine(" = - equal                  > - greater than");
1063     Console.WriteLine(" != - not equal           < - less than");
1064     Console.Write("Choice: ");
1065     string comparator = Console.ReadLine();
1066     Console.WriteLine();
1067
1068     // Get the user's search text and pipe that directly into the search method.
1069     Console.Write("Enter your search text: ");
1070     ItemDB foundItems = itemDB.Search(Console.ReadLine(),
1071         Enum.GetName(typeof(TypeMenu), type).ToLower(), field, comparator);
1072
1073     // Show the results.
1074     Console.WriteLine("");
1075     Console.WriteLine("-----");
1076     Console.WriteLine("| Search Results |");
1077     Console.WriteLine("-----");
1078     Console.WriteLine("{0} item{1} found.\n", foundItems.Count,
1079         foundItems.Count == 1 ? "" : "s");
1080
1081     // Display any found items.
1082     foundItems.DisplayAll();
1083
1084     // Return choice so the calling method knows what the choice was and can act
1085     // accordingly.
1086     return field;
1087 }
1088
1089 /*-----
1090  * Name:    DataStatistics
1091  * Type:    Method
1092  * Purpose: Display a count of unique Type fields and display those Type values.
1093  * Input:   Nothing.
1094  * Output:  Nothing.
1095  -----*/
1096 private void DataStatistics()
1097 {
1098     Console.WriteLine("-----");
1099     Console.WriteLine("| Statistics |");
1100     Console.WriteLine("-----");
1101
1102     itemDB.Statistics();
1103 }
1104
1105 /*-----
1106  * Name:    FieldMenuDisplay
1107  * Type:    Method
1108  * Purpose: Display the field menu and get a choice. Must have valid input to return.
1109  * Input:   Item item, used to get the user's choice of item type.
1110  * Output:  Item.FieldMenuHelper, representing the choice that was made.
1111  -----*/
1112 private Item.FieldMenuHelper FieldMenuDisplay(Item item)
1113 {
1114     TypeMenu type;
1115
1116     // Determine the type of the item.
1117     if (item is Business)
1118         type = TypeMenu.Business;
1119     else if (item is Park)
1120         type = TypeMenu.Park;
1121     else if (item is PublicFacility)
1122         type = TypeMenu.PublicFacility;

```

```
1123     else
1124         // Something went wrong; this should never be encountered.
1125         throw new InvalidOperationException(
1126             "Attempted to access field menu for an invalid object.");
1127
1128     // Call the full field menu and pass through the returned Item.FieldMenuHelper object.
1129     return FieldMenuDisplay(type);
1130 }
1131
1132 /*-----
1133 * Name:    FieldMenuDisplay
1134 * Type:    Method
1135 * Purpose: Display the field menu and get a choice. Must have valid input to return.
1136 * Input:   TypeMenu type, contains the user's choice of item type.
1137 * Output:  Item.FieldMenuHelper, representing the choice that was made.
1138 -----*/
1139 private Item.FieldMenuHelper FieldMenuDisplay(TypeMenu type)
1140 {
1141     Item.FieldMenuHelper menuChoice = 0;
1142     int offset = 0;
1143     bool invalid = true;
1144
1145     do
1146     {
1147         // Display the common part of the menu.
1148         Console.WriteLine("Please select the field you would like to work with:");
1149         Console.WriteLine(" 1) Name");
1150         Console.WriteLine(" 2) Type");
1151         Console.WriteLine(" 3) Street Address");
1152         Console.WriteLine(" 4) City");
1153         Console.WriteLine(" 5) State");
1154         Console.WriteLine(" 6) ZIP Code");
1155         Console.WriteLine(" 7) Latitude");
1156         Console.WriteLine(" 8) Longitude");
1157         Console.WriteLine(" 9) Phone Number");
1158
1159         if (type == TypeMenu.Business)
1160         {
1161             // Display the business-specific part of the menu.
1162             Console.WriteLine(" 10) Business License Fiscal Year");
1163             Console.WriteLine(" 11) Business License Number");
1164             Console.WriteLine(" 12) Business License Issued Date");
1165             Console.WriteLine(" 13) Business License Expiration Date");
1166             Console.WriteLine(" 14) Business License Status");
1167             Console.WriteLine(" 15) Council District");
1168             Console.WriteLine(" 16) Back");
1169
1170             // Offset used to convert choice to the proper FieldMenuHelper value.
1171             offset = Business.FieldOffset;
1172         }
1173         else if (type == TypeMenu.Park)
1174         {
1175             // Display the park-specific part of the menu.
1176             Console.WriteLine(" 10) Number of Baseball Diamonds");
1177             Console.WriteLine(" 11) Number of Basketball Courts");
1178             Console.WriteLine(" 12) Number of Golf Courses");
1179             Console.WriteLine(" 13) Number of Large Multipurpose FieldMenuHelper");
1180             Console.WriteLine(" 14) Number of Tennis Courts");
1181             Console.WriteLine(" 15) Number of Volleyball Courts");
1182             Console.WriteLine(" 16) Back");
1183
1184             // Offset used to convert choice to the proper FieldMenuHelper value.
1185             offset = Park.FieldOffset;
1186         }
1187     }
1188     else
1189     {
```

```

1189         // Display the public facility-specific part of the menu.
1190         Console.WriteLine(" 10) Back");
1191     }
1192
1193     // Ask the user for their choice.
1194     Console.Write("Choice: ");
1195     string input = Console.ReadLine();
1196
1197     // Extra line for formatting.
1198     Console.WriteLine();
1199
1200     // Attempts to parse user input, then adjusts menuChoice based on item type and
1201     // hands it off for actual validation.
1202     invalid = !Item.FieldMenuHelper.TryParse(input, out menuChoice);
1203     if (!invalid && menuChoice > Item.FieldCommonMax && type != TypeMenu.PublicFacility)
1204         menuChoice -= offset;
1205     invalid = invalid || !FieldMenuValidate(menuChoice, type);
1206 } while (invalid);
1207
1208 // Return the user's choice.
1209 return menuChoice;
1210 }
1211
1212 /*-----
1213  * Name:      FieldMenuValidate
1214  * Type:      Method
1215  * Purpose:   Validates that the choice by the user is within the limits and is logically
1216  *           possible.
1217  * Input:     Item.FieldMenuHelper value, contains the user's choice.
1218  * Input:     TypeMenu type, contains the user's choice of item type.
1219  * Output:    bool, representing whether the user's choice was valid or not.
1220 -----*/
1221 private bool FieldMenuValidate(Item.FieldMenuHelper value, TypeMenu type)
1222 {
1223     // General check to make sure that the user input is within valid limits.
1224     if (value < Item.FieldMin || value > Item.FieldMax)
1225         return false;
1226     // General check to see if the chosen field is one that is common to all items.
1227     else if (value >= Item.FieldCommonMin && value <= Item.FieldCommonMax)
1228         return true;
1229
1230     // Check whether the chosen field is valid for the given type.
1231     switch (type)
1232     {
1233     case TypeMenu.Business:
1234         if (value >= Business.FieldMin && value <= Business.FieldMax)
1235             return true;
1236         break;
1237     case TypeMenu.Park:
1238         if (value >= Park.FieldMin && value <= Park.FieldMax)
1239             return true;
1240         break;
1241     case TypeMenu.PublicFacility:
1242         if (value >= PublicFacility.FieldMin && value <= PublicFacility.FieldMax)
1243             return true;
1244         break;
1245     default:
1246         break;
1247     }
1248
1249     // Chosen field is not valid.
1250     return false;
1251 }
1252
1253 /*-----
1254  * Name:      MainMenuAction

```

```

1255     * Type:      Method
1256     * Purpose: Acts on the user's choice made at the Main Menu.
1257     * Input:   MainMenu choice, represents the action specified.
1258     * Output:  MainMenu, represents the action specified.
1259     -----*/
1260 private MainMenu MainMenuAction(MainMenu choice)
1261 {
1262     // Decide what to do based on the user's choice.
1263     switch (choice)
1264     {
1265         case MainMenu.Load:
1266             // Clear the ItemDB then load CSV files.
1267             DataLoad();
1268             break;
1269
1270         case MainMenu.Add:
1271             // Add a new item.
1272             DataAdd();
1273             break;
1274
1275         case MainMenu.Modify:
1276             // Modify an existing item.
1277             DataModify();
1278             break;
1279
1280         case MainMenu.Search:
1281             // Search items.
1282             DataSearch();
1283             break;
1284
1285         case MainMenu.Delete:
1286             // Delete an item.
1287             DataDelete();
1288             break;
1289
1290         case MainMenu.DisplayAll:
1291             // Display all the items.
1292             DataDisplayAll();
1293             break;
1294
1295         case MainMenu.Statistics:
1296             // Display
1297             DataStatistics();
1298             break;
1299
1300         case MainMenu.Exit:
1301             // Save then exit the program.
1302             DataSave();
1303             Console.WriteLine("Press any key to continue...");
1304             Console.ReadKey();
1305             break;
1306
1307         default:
1308             // Catch-all.
1309             break;
1310     }
1311
1312     // Return choice so the calling method knows what the choice was and can act accordingly.
1313     return choice;
1314 }
1315
1316 /*-----
1317  * Name:      MainMenuDisplay
1318  * Type:      Method
1319  * Purpose: Display the main menu and get a choice. Must have valid input to return.
1320  * Input:     Nothing.

```

```

1321     * Output: MainMenu, representing the choice that was made.
1322     -----*/
1323 private MainMenu MainMenuDisplay()
1324 {
1325     MainMenu menuChoice = 0;
1326     bool invalid = true;
1327
1328     do
1329     {
1330         // Display the menu.
1331         Console.WriteLine("-----");
1332         Console.WriteLine("| Main Interactive Menu |");
1333         Console.WriteLine("-----");
1334         Console.WriteLine("Please select an option:");
1335         Console.WriteLine(" 1) Clear List and Load Data");
1336         Console.WriteLine(" 2) Add New Item");
1337         Console.WriteLine(" 3) Modify Item");
1338         Console.WriteLine(" 4) Search Items");
1339         Console.WriteLine(" 5) Delete Item");
1340         Console.WriteLine(" 6) Display All Items");
1341         Console.WriteLine(" 7) Show Statistics");
1342         Console.WriteLine(" 8) Save and Exit");
1343         Console.Write("Choice: ");
1344
1345         // Get the user's choice.
1346         string input = Console.ReadLine();
1347
1348         // Extra line for formatting.
1349         Console.WriteLine();
1350
1351         // Validate the user input.
1352         invalid = !MainMenu.TryParse(input, out menuChoice) ||
1353                 !MainMenu.Validate(menuChoice);
1354     } while (invalid);
1355
1356     // Return the user's choice.
1357     return menuChoice;
1358 }
1359
1360 /*-----
1361  * Name:    MainMenuValidate
1362  * Type:    Method
1363  * Purpose: Validates that the choice by the user is within the limits and is logically
1364  *           possible.
1365  * Input:   MainMenu value, contains the user's choice.
1366  * Output:  bool, representing whether the user's choice was valid or not.
1367  -----*/
1368 private bool MainMenuValidate(MainMenu value)
1369 {
1370     // Check to make sure that the user input is within valid limits.
1371     if (value < MainMenu.Min || value > MainMenu.Max)
1372         return false;
1373
1374     // Otherwise, input is good.
1375     return true;
1376 }
1377
1378 /*-----
1379  * Name:    TypeMenuDisplay
1380  * Type:    Method
1381  * Purpose: Display the type menu and get a choice. Must have valid input to return.
1382  * Input:   Nothing.
1383  * Output:  TypeMenu, representing the choice that was made.
1384  -----*/
1385 private TypeMenu TypeMenuDisplay()
1386 {

```

```
1387         TypeMenu menuChoice = 0;
1388         bool invalid = true;
1389
1390         do
1391         {
1392             // Display the menu.
1393             Console.WriteLine("Please select an item type:");
1394             Console.WriteLine(" 1) Business");
1395             Console.WriteLine(" 2) Park");
1396             Console.WriteLine(" 3) Public Facility");
1397             Console.WriteLine(" 4) Back");
1398             Console.Write("Choice: ");
1399
1400             // Get the user's choice.
1401             string input = Console.ReadLine();
1402
1403             // Extra line for formatting.
1404             Console.WriteLine();
1405
1406             // Validate the user input.
1407             invalid = !TypeMenu.TryParse(input, out menuChoice) ||
1408                 !TypeMenu.Validate(menuChoice);
1409         } while (invalid);
1410
1411         // Return the user's choice.
1412         return menuChoice;
1413     }
1414
1415     /*-----
1416     * Name:      TypeMenu.Validate
1417     * Type:      Method
1418     * Purpose:  Validates that the choice by the user is within the limits and is logically
1419     *           possible.
1420     * Input:    TypeMenu value, contains the user's choice.
1421     * Output:   bool, representing whether the user's choice was valid or not.
1422     -----*/
1423     private bool TypeMenu.Validate(TypeMenu value)
1424     {
1425         // Check to make sure that the user input is within valid limits.
1426         if (value < TypeMenu.Min || value > TypeMenu.Max)
1427             return false;
1428
1429         // Otherwise, input is good.
1430         return true;
1431     }
1432 }
1433 }
1434
```

```

1  /*-----
2  * Author:      Dan Cassidy and Dr. Raman Adaikkalavan
3  * Date:        2015-06-17
4  * Assignment:  cView-P3
5  * Source File: ItemDB.cs
6  * Language:    C#
7  * Course:      CSCI-C 490, C# Programming, MoWe 08:00
8  * Purpose:     Encapsulates a List-based collection of Item objects and contains related methods
9  *              and properties.
10 -----*/
11
12 using System;
13 using System.Collections;
14 using System.Collections.Generic;
15 using System.Linq;
16 using System.Text;
17 using System.Threading.Tasks;
18
19 namespace Ph3
20 {
21     public class ItemDB : IEnumerable
22     {
23         /*-----
24         * Name:      KeyStart
25         * Type:      Constant
26         * Purpose:   Contains the default value for currentItemKey.
27         -----*/
28         private const int KeyStart = 1;
29
30         /*-----
31         * Name:      currentItemKey
32         * Type:      Field
33         * Purpose:   Implements a counter for the ID number for Item objects. This is due to the fact
34         *             that itemList.Count becomes unreliable if objects are removed from the list.
35         -----*/
36         private static int currentItemKey = KeyStart;
37
38         /*-----
39         * Name:      itemList
40         * Type:      Field
41         * Purpose:   List of Item objects.
42         -----*/
43         private List<Item> itemList = new List<Item>();
44
45         /*-----
46         * Name:      Count
47         * Type:      Property
48         * Purpose:   Enable access to the Count property of itemList.
49         -----*/
50         public int Count { get { return itemList.Count; } }
51
52         /*-----
53         * Name:      IsChanged
54         * Type:      Property
55         * Purpose:   Flag that determines whether the itemDB has been modified (true) or not (false).
56         -----*/
57         public bool IsChanged { get; private set; }
58
59         /*-----
60         * Name:      Add
61         * Type:      Method
62         * Purpose:   Add the specified item object to the ItemDB object.
63         * Input:     Item item, specifies the item to be added to the ItemDB object.
64         * Output:    Nothing.
65         -----*/
66         public void Add(Item item)

```



```

67     {
68         // Set the item ID to whatever the current key is, increment the key, then add the item.
69         item.ItemID = currentItemKey++;
70         itemList.Add(item);
71         IsChanged = true;
72     }
73
74     /*-----
75     * Name:      Delete
76     * Type:      Method
77     * Purpose: Attempt to delete the Item with the the specified ItemID.
78     * Input:   int itemIDToDelete, specifies the ItemID to delete.
79     * Output:  bool, represents whether the deletion was successful or not.
80     -----*/
81     public bool Delete(int itemIDToDelete)
82     {
83         try
84         {
85             itemList.RemoveAt(GetItemIndex(itemIDToDelete));
86             IsChanged = true;
87             return true;
88         }
89         catch (Exception ex)
90         {
91             Console.WriteLine(ex.Message);
92             return false;
93         }
94     }
95
96     /*-----
97     * Name:      DisplayAll
98     * Type:      Method
99     * Purpose: Display a paginated list of all the items in the ItemDB object. Can be a
100    * simplified list or not.
101    * Input:   bool simplified, tells the method whether it should display simplified listing
102    *          or not.
103    * Output:  Nothing.
104    -----*/
105    public void DisplayAll(bool simplified = false)
106    {
107        // Helper constants to determine how many lines are going to be used for displaying each
108        // type of item.
109        const int linesDisplayedPerBusiness = 10;
110        const int linesDisplayedPerPark = 12;
111        const int linesDisplayedPerPublicFacility = 6;
112        const int linesDisplayedSimplified = 4;
113
114        // Variables to help with controlling pagination flow.
115        bool displayAll = false;
116        int linesToBeDisplayed = 0;
117        int linesDisplayed = 0;
118        bool validInput = false;
119        ConsoleKeyInfo keyPress;
120
121        foreach (var item in itemList)
122        {
123            // If the user has chosen to display everything, don't both with the other logic.
124            if (!displayAll)
125            {
126                // Figure out how many lines are about to be displayed.
127                if (simplified)
128                    linesToBeDisplayed = linesDisplayedSimplified;
129                else if (item.ItemType == "business")
130                    linesToBeDisplayed = linesDisplayedPerBusiness;
131                else if (item.ItemType == "park")
132                    linesToBeDisplayed = linesDisplayedPerPark;

```

```

133         else if (item.ItemType == "publicfacility")
134             linesToBeDisplayed = linesDisplayedPerPublicFacility;
135
136         // If the number of lines about to be displayed will put the displayed number of
137         // lines since last reset at greater than the number of lines available for
138         // display, pause output and ask the user what to do.
139         if (linesDisplayed + linesToBeDisplayed >= Console.WindowHeight - 1)
140         {
141             Console.WriteLine("Enter for next item. Space for next page. " +
142                 "Ctrl+Enter for all. Esc to abort.\n");
143             do
144             {
145                 // Reset valid input flag and read user input.
146                 validInput = false;
147                 keyPress = Console.ReadKey(true);
148
149                 switch (keyPress.Key)
150                 {
151                     case ConsoleKey.Escape:
152                         if (keyPress.Modifiers == 0)
153                             // User pressed Escape key; abort display method.
154                             return;
155                         break;
156
157                     case ConsoleKey.Spacebar:
158                         if (keyPress.Modifiers == 0)
159                         {
160                             // User wishes to display another page; reset the number of
161                             // lines displayed to 0.
162                             linesDisplayed = 0;
163                             validInput = true;
164                         }
165                         break;
166
167                     case ConsoleKey.Enter:
168                         if (keyPress.Modifiers == ConsoleModifiers.Control)
169                         {
170                             // User wishes to display everything.
171                             displayAll = true;
172                             validInput = true;
173                         }
174                         else if (keyPress.Modifiers == 0)
175                         {
176                             // User wishes to display only the next item.
177                             linesDisplayed -= linesToBeDisplayed;
178                             validInput = true;
179                         }
180                         break;
181
182                     default:
183                         break;
184                 }
185                 // Loop while the user has not provided valid input.
186                 } while (!validInput);
187             }
188             // Update the number of lines that have been displayed.
189             linesDisplayed += linesToBeDisplayed;
190         }
191         // Display the item. Must use the ToString() method, otherwise VS complains that
192         // there are no implicit conversions between Item and string types.
193         Console.WriteLine("{0}\n", simplified ? item.ToStringSimple() : item.ToString());
194     }
195
196     if (itemList.Count == 0)
197         Console.WriteLine("No items to display.\n");
198 }

```

```

199
200 /*-----
201  * Name:    GetItem
202  * Type:    Method
203  * Purpose: Get a copy of the item with the specified ItemID.
204  * Input:   int itemID, the itemID of the item to get.
205  * Output:  Item, contains a copy of the object with itemID, or null if not found.
206  -----*/
207 public Item GetItem(int itemID)
208 {
209     Item tempItem = itemList.Find(i => i.ItemID == itemID);
210
211     if (tempItem is Business)
212         return new Business(tempItem as Business);
213     else if (tempItem is Park)
214         return new Park(tempItem as Park);
215     else if (tempItem is PublicFacility)
216         return new PublicFacility(tempItem as PublicFacility);
217     else
218         return null;
219 }
220
221 /*-----
222  * Name:    GetItemIndex
223  * Type:    Method
224  * Purpose: Finds the index of the specified item ID.
225  * Input:   int itemID, contains the item ID to search for.
226  * Output:  int, contains the index where the item ID can be found.
227  -----*/
228 public int GetItemIndex(int itemID)
229 {
230     return itemList.FindIndex(i => i.ItemID == itemID);
231 }
232
233 /*-----
234  * Name:    Modify
235  * Type:    Method
236  * Purpose: Modifies an Item in the list.
237  * Input:   Item item, contains the item that will be matched with and replace the item with
238  *          the same ItemID.
239  * Output:  Nothing.
240  -----*/
241 public void Modify(Item item)
242 {
243     int index = GetItemIndex(item.ItemID);
244
245     // Verify that the ItemID is in the list and that ItemType is the same.
246     if (index >= 0 && itemList[index].ItemType == item.ItemType)
247         // Verify that the items are of the same type.
248         if ((itemList[index] is Business && item is Business) ||
249             (itemList[index] is Park && item is Park) ||
250             (itemList[index] is PublicFacility && item is PublicFacility))
251             {
252                 // Replace the item reference in the list.
253                 itemList[index] = item;
254                 IsChanged = true;
255             }
256 }
257
258 /*-----
259  * Name:    Reset
260  * Type:    Method
261  * Purpose: Clears the ItemDB, resets currentItemKey, and resets IsChanged.
262  * Input:   Nothing.
263  * Output:  Nothing.
264  -----*/

```

```

265     public void Reset()
266     {
267         itemList.Clear();
268         currentItemKey = KeyStart;
269         IsChanged = false;
270     }
271
272     /*-----
273     * Name:      Search
274     * Type:      Method
275     * Purpose:   Performs a search based on the comparator on the specified item type and field.
276     * Input:     string toSearchFor, contains the string that is being searched for.
277     * Input:     string itemType, contains the item type to search through.
278     * Input:     Item.FieldMenuHelper field, contains the field to search through.
279     * Input:     string comparator, contains the comparator that will be used. Valid choices are
280     *             !=, =, <=, >=, <, >, and !|. Everything else does a "contains"-style search.
281     * Output:    ItemDB object that contains the results of the search.
282     -----*/
283     public ItemDB Search(string toSearchFor, string itemType, Item.FieldMenuHelper field,
284         string comparator = "")
285     {
286         if (itemList.Count == 0)
287             return this;
288
289         var ignoreCase = StringComparison.OrdinalIgnoreCase;
290
291         // Create base list and object for ease-of-use inside the switch.
292         var typeLimitedList = this.itemList.Where(i => i.ItemType == itemType);
293         object baseObject = typeLimitedList.Select(i => i[field]).First();
294
295         switch (comparator)
296         {
297             case "!=":
298                 if (baseObject is DateTime)
299                     return new ItemDB() { itemList = typeLimitedList.
300                         Where(i => (DateTime)i[field] != SimpleConvert.ToDateTime(toSearchFor)).
301                         ToList() };
302                 else if (baseObject is float)
303                     return new ItemDB() { itemList = typeLimitedList.
304                         Where(i => (float)i[field] != SimpleConvert.ToSingle(toSearchFor)).
305                         ToList() };
306                 else if (baseObject is int)
307                     return new ItemDB() { itemList = typeLimitedList.
308                         Where(i => (int)i[field] != SimpleConvert.ToInt32(toSearchFor)).
309                         ToList() };
310                 else
311                     return new ItemDB() { itemList = typeLimitedList.
312                         Where(i => (string)i[field] != toSearchFor).
313                         ToList() };
314             case "=":
315                 if (baseObject is DateTime)
316                     return new ItemDB() { itemList = typeLimitedList.
317                         Where(i => (DateTime)i[field] == SimpleConvert.ToDateTime(toSearchFor)).
318                         ToList() };
319                 else if (baseObject is float)
320                     return new ItemDB() { itemList = typeLimitedList.
321                         Where(i => (float)i[field] == SimpleConvert.ToSingle(toSearchFor)).
322                         ToList() };
323                 else if (baseObject is int)
324                     return new ItemDB() { itemList = typeLimitedList.
325                         Where(i => (int)i[field] == SimpleConvert.ToInt32(toSearchFor)).
326                         ToList() };
327                 else
328                     return new ItemDB() { itemList = typeLimitedList.
329                         Where(i => (string)i[field] == toSearchFor).
330

```

```

331         ToList() };
332
333     case "<=":
334         if (baseObject is DateTime)
335             return new ItemDB() { itemList = typeLimitedList.
336                 Where(i => (DateTime)i[field] <= SimpleConvert.ToDateTime(toSearchFor)).
337                 ToList() };
338         else if (baseObject is float)
339             return new ItemDB() { itemList = typeLimitedList.
340                 Where(i => (float)i[field] <= SimpleConvert.ToSingle(toSearchFor)).
341                 ToList() };
342         else if (baseObject is int)
343             return new ItemDB() { itemList = typeLimitedList.
344                 Where(i => (int)i[field] <= SimpleConvert.ToInt32(toSearchFor)).
345                 ToList() };
346         else
347             Console.WriteLine(
348                 "That comparator doesn't work for this field. Switching to |.");
349         break;
350
351     case ">=":
352         if (baseObject is DateTime)
353             return new ItemDB() { itemList = typeLimitedList.
354                 Where(i => (DateTime)i[field] >= SimpleConvert.ToDateTime(toSearchFor)).
355                 ToList() };
356         else if (baseObject is float)
357             return new ItemDB() { itemList = typeLimitedList.
358                 Where(i => (float)i[field] >= SimpleConvert.ToSingle(toSearchFor)).
359                 ToList() };
360         else if (baseObject is int)
361             return new ItemDB() { itemList = typeLimitedList.
362                 Where(i => (int)i[field] >= SimpleConvert.ToInt32(toSearchFor)).
363                 ToList() };
364         else
365             Console.WriteLine(
366                 "That comparator doesn't work for this field. Switching to |.");
367         break;
368
369     case "<":
370         if (baseObject is DateTime)
371             return new ItemDB() { itemList = typeLimitedList.
372                 Where(i => (DateTime)i[field] < SimpleConvert.ToDateTime(toSearchFor)).
373                 ToList() };
374         else if (baseObject is float)
375             return new ItemDB() { itemList = typeLimitedList.
376                 Where(i => (float)i[field] < SimpleConvert.ToSingle(toSearchFor)).
377                 ToList() };
378         else if (baseObject is int)
379             return new ItemDB() { itemList = typeLimitedList.
380                 Where(i => (int)i[field] < SimpleConvert.ToInt32(toSearchFor)).
381                 ToList() };
382         else
383             Console.WriteLine(
384                 "That comparator doesn't work for this field. Switching to |.");
385         break;
386
387     case ">":
388         if (baseObject is DateTime)
389             return new ItemDB() { itemList = typeLimitedList.
390                 Where(i => (DateTime)i[field] > SimpleConvert.ToDateTime(toSearchFor)).
391                 ToList() };
392         else if (baseObject is float)
393             return new ItemDB() { itemList = typeLimitedList.
394                 Where(i => (float)i[field] > SimpleConvert.ToSingle(toSearchFor)).
395                 ToList() };
396         else if (baseObject is int)

```

```

397         return new ItemDB() { itemList = typeLimitedList.
398             Where(i => (int)i[field] > SimpleConvert.ToInt32(toSearchFor)).
399             ToList() };
400     else
401         Console.WriteLine(
402             "That comparator doesn't work for this field. Switching to |.");
403     break;
404
405     case "||":
406         return new ItemDB() { itemList = typeLimitedList.
407             Where(i => i[field].ToString().IndexOf(toSearchFor, ignoreCase) < 0).
408             ToList() };
409
410     default:
411         break;
412 }
413
414 // Default/catch-all search.
415 return new ItemDB() { itemList = typeLimitedList.
416     Where(i => i[field].ToString().IndexOf(toSearchFor, ignoreCase) >= 0).
417     ToList() };
418 }
419
420 /*-----
421  * Name:    Statistics
422  * Type:    Method
423  * Purpose: Displays the number of unique Type fields, and then displays the field values
424  *           and their count.
425  * Input:   Nothing.
426  * Output:  Nothing.
427  -----*/
428 public void Statistics()
429 {
430     // Create a Dictionary to keep track of the unique Item.Type values.
431     Dictionary<string, int> types = new Dictionary<string, int>();
432
433     // Get a sorted lowercase list of unique Item.Type values and Add the aforementioned
434     // list to the dictionary.
435     var uniqueTypes = itemList.Select(i => i.Type.ToLower()).Distinct().OrderBy(s => s);
436     foreach (var type in uniqueTypes)
437         types.Add(type, 0);
438
439     // Run through the list and increment the count of any type when it is encountered, then
440     // display all the results.
441     foreach (var item in itemList)
442         types[item.Type.ToLower()]++;
443     Console.WriteLine("{0} unique type{1} of item{1} found.\n", types.Count,
444         types.Count != 1 ? "s" : "");
445     foreach (var type in types)
446         Console.WriteLine("{0}: {1}", type.Key, type.Value);
447     if (types.Count > 0)
448         Console.WriteLine();
449 }
450
451 // Implementation for the GetEnumerator method. Source:
452 // https://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.110).aspx
453 IEnumerator IEnumerable.GetEnumerator()
454 {
455     return (IEnumerator)GetEnumerator();
456 }
457
458 public ItemDBEnum GetEnumerator()
459 {
460     return new ItemDBEnum(itemList);
461 }
462 }

```

```
463
464 public class ItemDBEnum : IEnumerator
465 {
466     // Enumerator for the ItemDB class. Much help came from MSDN.
467     // https://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.110).aspx
468
469     private List<Item> itemList;
470
471     int position = -1;
472
473     public ItemDBEnum(List<Item> list)
474     {
475         itemList = list;
476     }
477
478     object IEnumerator.Current
479     {
480         get
481         {
482             return Current;
483         }
484     }
485
486     public Item Current
487     {
488         get
489         {
490             try
491             {
492                 return itemList[position];
493             }
494             catch (IndexOutOfRangeException)
495             {
496                 throw new InvalidOperationException();
497             }
498         }
499     }
500
501     public bool MoveNext()
502     {
503         position++;
504         return (position < itemList.Count);
505     }
506
507     public void Reset()
508     {
509         position = -1;
510     }
511 }
512 }
513
```

```

1  /*-----
2  * Author:      Dan Cassidy and Dr. Raman Adaikkalavan
3  * Date:        2015-06-17
4  * Assignment:  cView-P3
5  * Source File: Item.cs
6  * Language:    C#
7  * Course:      CSCI-C 490, C# Programming, MoWe 08:00
8  * Purpose:     Provides the base abstract class for data items along with some supporting methods.
9  *-----*/
10
11 using System;
12 using System.Collections.Generic;
13 using System.Linq;
14 using System.Runtime.CompilerServices;
15 using System.Text;
16 using System.Threading.Tasks;
17
18 namespace Ph3
19 {
20     public abstract class Item
21     {
22         /*-----
23         * Type:      Helper Constants
24         *-----*/
25         public const FieldMenuHelper FieldCommonMin = FieldMenuHelper.Name;
26         public const FieldMenuHelper FieldCommonMax = FieldMenuHelper.Phone;
27         public const FieldMenuHelper FieldMin = FieldMenuHelper.Name;
28         public const FieldMenuHelper FieldMax = FieldMenuHelper.BackPark;
29
30         /*-----
31         * Type:      Constructor
32         * Purpose:    Basic no-parameter constructor.
33         * Input:      Nothing.
34         *-----*/
35         public Item()
36         {
37             // Nothing else to do.
38         }
39
40         /*-----
41         * Type:      Constructor
42         * Purpose:    Copy constructor.
43         * Input:      Item fromItem, reference to the other Item from which fields should be copied.
44         *-----*/
45         public Item(Item fromItem)
46         {
47             ItemID = fromItem.ItemID;
48
49             Name = fromItem.Name;
50             Type = fromItem.Type;
51             StreetAddress = fromItem.StreetAddress;
52             City = fromItem.City;
53             State = fromItem.State;
54             Zip = fromItem.Zip;
55             Latitude = fromItem.Latitude;
56             Longitude = fromItem.Longitude;
57             Phone = fromItem.Phone;
58         }
59
60         /*-----
61         * Type:      Constructor
62         * Purpose:    Constructor that will fill all the properties except ItemID and ItemType.
63         * Input:      string name, contains the desired Name for the object.
64         * Input:      string type, contains the desired Type for the object.
65         * Input:      string streetAddress, contains the desired StreetAddress for the object.
66         * Input:      string city, contains the desired City for the object.

```



```

67      * Input:   string state, contains the desired State for the object.
68      * Input:   string zip, contains the desired Zip for the object.
69      * Input:   string latitude, contains the desired Latitude for the object.
70      * Input:   string longitude, contains the desired Longitude for the object.
71      * Input:   string phone, contains the desired Phone for the object.
72      -----*/
73      public Item(string name, string type, string streetAddress, string city, string state,
74                  string zip, string latitude, string longitude, string phone)
75      {
76          Name = name;
77          Type = type;
78          StreetAddress = streetAddress;
79          City = city;
80          State = state;
81          Zip = zip;
82          Latitude = latitude;
83          Longitude = longitude;
84          Phone = phone;
85      }
86
87      /*-----
88      * Name:      FieldMenuHelper
89      * Type:      Enum
90      * Purpose: Represents the fields in use in this class, with additions for its derived
91      *           classes.
92      -----*/
93      public enum FieldMenuHelper
94      {
95          // Common Fields
96          Name = 1,
97          Type,
98          StreetAddress,
99          City,
100         State,
101         Zip,
102         Latitude,
103         Longitude,
104         Phone,
105         Back,
106
107         // Business Fields
108         LicenseFiscalYear,
109         LicenseNumber,
110         LicenseIssueDate,
111         LicenseExpirDate,
112         LicenseStatus,
113         CouncilDistrict,
114         BackBusiness,
115
116         // Park Fields
117         FeatureBaseball,
118         FeatureBasketball,
119         FeatureGolf,
120         FeatureLargeMPField,
121         FeatureTennis,
122         FeatureVolleyball,
123         BackPark
124     }
125
126     /*-----
127     * BEGIN UNTOUCHABLE CODE -->
128     -----*/
129     // Create an ID for each item. So if you have 10 parks and 5 businesses, ID will be 1 to 15.
130     public abstract int ItemID { get; set; }
131
132     // Value will be "business", "park", or "publicfacility".

```

```

133     public abstract string ItemType { get; set; }
134
135     // Populate from CSV
136     public abstract string Name { get; set; }
137     public abstract string Type { get; set; }
138     public abstract string StreetAddress { get; set; }
139     public abstract string City { get; set; }
140     public abstract string State { get; set; }
141     public abstract string Zip { get; set; }
142     public abstract string Latitude { get; set; }
143     public abstract string Longitude { get; set; }
144     public abstract string Phone { get; set; }
145     /*-----
146     * <-- END UNTOUCHABLE CODE
147     -----*/
148
149     /*-----
150     * Name:      this[]
151     * Type:      Indexer
152     * Purpose:   Provides easy access to the properties of the class. Need to change the indexer
153     *            name because its default is "Item" and the compiler throws a fit because the
154     *            class is already named that.
155     * Input:     FieldMenuHelper fiendNum, represents the desired property.
156     * Output:    object, contains whichever property was desired, or 0 if the property was not
157     *            found.
158     -----*/
159     [IndexerName("Index")]
160     public virtual object this[FieldMenuHelper fieldNum]
161     {
162         get
163         {
164             switch (fieldNum)
165             {
166                 case FieldMenuHelper.Name:
167                     return Name;
168                 case FieldMenuHelper.Type:
169                     return Type;
170                 case FieldMenuHelper.StreetAddress:
171                     return StreetAddress;
172                 case FieldMenuHelper.City:
173                     return City;
174                 case FieldMenuHelper.State:
175                     return State;
176                 case FieldMenuHelper.Zip:
177                     return Zip;
178                 case FieldMenuHelper.Latitude:
179                     return Latitude;
180                 case FieldMenuHelper.Longitude:
181                     return Longitude;
182                 case FieldMenuHelper.Phone:
183                     return Phone;
184                 default:
185                     return 0;
186             }
187         }
188     }
189
190     /*-----
191     * Name:      ToStringCSV
192     * Type:      Method
193     * Purpose:   Serializes the data contained in the object into a comma-separated value string.
194     * Input:     Nothing.
195     * Output:    string, representing the data of this object as serialized to a CSV string.
196     -----*/
197     public virtual string ToStringCSV()
198     {

```

```

199         char separator = ',';
200         return Name + separator + Type + separator + StreetAddress + separator + City +
201             separator + State + separator + Zip + separator + Latitude + separator + Longitude +
202             separator + Phone;
203     }
204
205     /*-----
206     * Name: ToStringSimple
207     * Type: Method
208     * Purpose: Formats the data contained in the object into a simplified string containing
209     *           only the ItemID, ItemType, and Name properties.
210     * Input: Nothing.
211     * Output: string, representing a simplified view of this object.
212     -----*/
213     public virtual string ToStringSimple()
214     {
215         // Returns a string formatted as follows:
216         // Item ID: <ItemID>
217         // Item Type: <ItemType>
218         // Name: <Name>
219         return string.Format(
220             " Item ID: {0}\n" +
221             "Item Type: {1}\n" +
222             " Name: {2}",
223             ItemID,
224             ItemType,
225             Name);
226     }
227 }
228 }
229

```

```

1  /*-----
2  * Author:      Dan Cassidy
3  * Date:        2015-06-17
4  * Assignment:  cView-P3
5  * Source File: Business.cs
6  * Language:    C#
7  * Course:      CSCI-C 490, C# Programming, MoWe 08:00
8  * Purpose:     Contains the Business class, derived from the Item abstract class, and supporting
9  *              methods.
10 -----*/
11
12 using System;
13 using System.Collections.Generic;
14 using System.Linq;
15 using System.Text;
16 using System.Threading.Tasks;
17
18 namespace Ph3
19 {
20     public class Business : Item
21     {
22         /*-----
23         * Type:      Helper Constants
24         -----*/
25         public new const FieldMenuHelper FieldMin = FieldMenuHelper.LicenseFiscalYear;
26         public new const FieldMenuHelper FieldMax = FieldMenuHelper.BackBusiness;
27         public const int FieldOffset = 1;
28
29         /*-----
30         * Type:      Private Fields
31         -----*/
32         private string itemType = "business";
33         private int licenseFiscalYear;
34         private int licenseNumber;
35
36         /*-----
37         * Type:      Constructor
38         * Purpose:   Basic no-parameter constructor.
39         * Input:     Nothing.
40         -----*/
41         public Business()
42         {
43             // Nothing else to do.
44         }
45
46         /*-----
47         * Type:      Constructor
48         * Purpose:   Copy constructor.
49         * Input:     Business fromItem, reference to the other Business from which fields should be
50         *              copied.
51         -----*/
52         public Business(Business fromItem)
53             : base(fromItem)
54         {
55             itemType = fromItem.itemType;
56
57             LicenseFiscalYear = fromItem.LicenseFiscalYear;
58             LicenseNumber = fromItem.LicenseNumber;
59             LicenseIssueDate = fromItem.LicenseIssueDate;
60             LicenseExpirDate = fromItem.LicenseExpirDate;
61             LicenseStatus = fromItem.LicenseStatus;
62             CouncilDistrict = fromItem.CouncilDistrict;
63         }
64
65         /*-----
66         * Type:      Constructor

```

```

67      * Purpose: Constructor that will fill all the properties except ItemID and ItemType.
68      * Input:   string name, contains the desired Name for the object.
69      * Input:   string type, contains the desired Type for the object.
70      * Input:   string streetAddress, contains the desired StreetAddress for the object.
71      * Input:   string city, contains the desired City for the object.
72      * Input:   string state, contains the desired State for the object.
73      * Input:   string zip, contains the desired Zip for the object.
74      * Input:   string latitude, contains the desired Latitude for the object.
75      * Input:   string longitude, contains the desired Longitude for the object.
76      * Input:   string phone, contains the desired Phone for the object.
77      * Input:   int licenseFiscalYear, contains the desired LicenseFiscalYear for the object.
78      * Input:   int licenseNumber, contains the desired LicenseNumber for the object.
79      * Input:   DateTime licenseIssueDate, contains the desired LicenseIssueDate for the object.
80      * Input:   DateTime licenseExpirDate, contains the desired LicenseExpirDate for the object.
81      * Input:   string licenseStatus, contains the desired LicenseStatus for the object.
82      * Input:   string councilDistrict, contains the desired CouncilDistrict for the object.
83      * Output:  Nothing.
84      -----*/
85      public Business(string name, string type, string streetAddress, string city, string state,
86                      string zip, string latitude, string longitude, string phone, int licenseFiscalYear,
87                      int licenseNumber, DateTime licenseIssueDate, DateTime licenseExpirDate,
88                      string licenseStatus, string councilDistrict)
89      {
90          : base(name, type, streetAddress, city, state, zip, latitude, longitude, phone)
91      {
92          LicenseFiscalYear = licenseFiscalYear;
93          LicenseNumber = licenseNumber;
94          LicenseIssueDate = licenseIssueDate;
95          LicenseExpirDate = licenseExpirDate;
96          LicenseStatus = licenseStatus;
97          CouncilDistrict = councilDistrict;
98      }
99      /*-----
100      * Type:      Auto-implemented Properties
101      -----*/
102      public override int ItemID { get; set; }           // Item ID
103
104      public override string Name { get; set; }         // Business Name
105      public override string Type { get; set; }         // Business Classification
106      public override string StreetAddress { get; set; } // Street Address
107      public override string City { get; set; }         // City
108      public override string State { get; set; }        // State
109      public override string Zip { get; set; }          // Zip Code
110      public override string Latitude { get; set; }     // Latitude
111      public override string Longitude { get; set; }    // Longitude
112      public override string Phone { get; set; }        // Phone Number
113
114      public DateTime LicenseIssueDate { get; set; }    // Business License Issue Date
115      public DateTime LicenseExpirDate { get; set; }    // Business License Expiration Date
116      public string LicenseStatus { get; set; }         // Business License Status
117      public string CouncilDistrict { get; set; }       // Council District
118
119      /*-----
120      * Name:      ItemType
121      * Type:      Property
122      * Purpose: Provides access to the itemType field.
123      -----*/
124      public override string ItemType
125      {
126          get
127          {
128              return itemType;
129          }
130          set
131          {
132              // Do nothing.

```

```

133     }
134 }
135
136 /*-----
137  * Name:    LicenseFiscalYear
138  * Type:    Property
139  * Purpose: Provides access to the licenseFiscalYear field, and validation for the same.
140 -----*/
141 public int LicenseFiscalYear
142 {
143     get
144     {
145         return licenseFiscalYear;
146     }
147     set
148     {
149         if (value >= 0)
150             licenseFiscalYear = value;
151     }
152 }
153
154 /*-----
155  * Name:    LicenseNumber
156  * Type:    Property
157  * Purpose: Provides access to the licenseNumber field, and validation for the same.
158 -----*/
159 public int LicenseNumber
160 {
161     get
162     {
163         return licenseNumber;
164     }
165     set
166     {
167         if (value >= 0)
168             licenseNumber = value;
169     }
170 }
171
172 /*-----
173  * Name:    this[]
174  * Type:    Indexer
175  * Purpose: Provides easy access to the properties of the class.
176  * Input:   FieldMenuHelper fiendNum, represents the desired property.
177  * Output:  object, contains whichever property was desired, or 0 if the property was not
178  *          found.
179 -----*/
180 public override object this[FieldMenuHelper fieldNum]
181 {
182     get
183     {
184         switch (fieldNum)
185         {
186             case FieldMenuHelper.LicenseFiscalYear:
187                 return LicenseFiscalYear;
188             case FieldMenuHelper.LicenseNumber:
189                 return LicenseNumber;
190             case FieldMenuHelper.LicenseIssueDate:
191                 return LicenseIssueDate;
192             case FieldMenuHelper.LicenseExpirDate:
193                 return LicenseExpirDate;
194             case FieldMenuHelper.LicenseStatus:
195                 return LicenseStatus;
196             case FieldMenuHelper.CouncilDistrict:
197                 return CouncilDistrict;
198             default:

```

```

199         return base[fieldNum];
200     }
201 }
202
203
204 /*-----
205  * Name: ToStringCSV
206  * Type: Method
207  * Purpose: Serializes the data contained in the object into a comma-separated value string.
208  * Input: Nothing.
209  * Output: string, representing the data of this object as serialized to a CSV string.
210 -----*/
211 public override string ToStringCSV()
212 {
213     char separator = ',';
214     return base.ToStringCSV() + separator + LicenseFiscalYear + '-' + LicenseNumber +
215         separator + LicenseIssueDate.ToShortDateString() + separator +
216         LicenseExpirDate.ToShortDateString() + separator + LicenseStatus + separator +
217         CouncilDistrict;
218 }
219
220 /*-----
221  * Name: ToString
222  * Type: Method
223  * Purpose: Override of ToString() method. Formats the data contained in this object so it
224  * looks pretty.
225  * Input: Nothing.
226  * Output: string, containing serialized object data.
227 -----*/
228 public override string ToString()
229 {
230     // Returns a string formatted as follows:
231     // Item ID (Item Type): <ItemID> (<ItemType>)
232     // Business Name (Type): <Name> (<Type>)
233     // Address: <StreetAddress>, <City>, <State> <Zip>
234     // GPS Coordinates: (<Latitude>, <Longitude>)
235     // Phone Number: <Phone>
236     // License Number: <LicenseFiscalYear>-<LicenseNumber>
237     // Valid: From <LicenseIssueDate> to <LicenseExpirDate>
238     // Status: <LicenseStatus>
239     // Council District: <CouncilDistrict>
240     return string.Format(
241         " Item ID (Item Type): {0} ({1})\n" +
242         "Business Name (Type): {2} ({3})\n" +
243         " Address: {4}, {5}, {6} {7}\n" +
244         " GPS Coordinates: ({8}, {9})\n" +
245         " Phone Number: {10}\n" +
246         " License Number: {11}-{12}\n" +
247         " Valid: From {13} to {14}\n" +
248         " Status: {15}\n" +
249         " Council District: {16}",
250         ItemID, ItemType,
251         Name, Type,
252         StreetAddress, City, State, Zip,
253         Latitude, Longitude,
254         Phone,
255         LicenseFiscalYear, LicenseNumber,
256         LicenseIssueDate.ToShortDateString(), LicenseExpirDate.ToShortDateString(),
257         LicenseStatus,
258         CouncilDistrict);
259 }
260 }
261 }
262

```

```

1  /*-----
2  * Author:      Dan Cassidy and Dr. Raman Adaikkalavan
3  * Date:        2015-06-17
4  * Assignment:  cView-P3
5  * Source File: Park.cs
6  * Language:    C#
7  * Course:      CSCI-C 490, C# Programming, MoWe 08:00
8  * Purpose:     Contains the Park class, derived from the Item abstract class, and supporting
9  *              methods.
10 -----*/
11
12 using System;
13 using System.Collections.Generic;
14 using System.Linq;
15 using System.Text;
16 using System.Threading.Tasks;
17
18 namespace Ph3
19 {
20     public class Park : Item
21     {
22         /*-----
23         * Type:      Helper Constants
24         -----*/
25         public new const FieldMenuHelper FieldMin = FieldMenuHelper.FeatureBaseball;
26         public new const FieldMenuHelper FieldMax = FieldMenuHelper.BackPark;
27         public const int FieldOffset = 8;
28
29         /*-----
30         * Type:      Private Fields
31         -----*/
32         private string itemType = "park";
33         private int featureBaseball;
34         private float featureBasketball;
35         private float featureGolf;
36         private int featureLargeMPField;
37         private int featureTennis;
38         private int featureVolleyball;
39
40         /*-----
41         * Type:      Constructor
42         * Purpose:   Basic no-parameter constructor.
43         * Input:     Nothing.
44         -----*/
45         public Park()
46         {
47             // Nothing else to do.
48         }
49
50         /*-----
51         * Type:      Constructor
52         * Purpose:   Copy constructor.
53         * Input:     Park fromItem, reference to the other Park from which fields should be copied.
54         -----*/
55         public Park(Park fromItem)
56             : base(fromItem)
57         {
58             itemType = fromItem.itemType;
59
60             FeatureBaseball = fromItem.FeatureBaseball;
61             FeatureBasketball = fromItem.FeatureBasketball;
62             FeatureGolf = fromItem.FeatureGolf;
63             FeatureLargeMPField = fromItem.FeatureLargeMPField;
64             FeatureTennis = fromItem.FeatureTennis;
65             FeatureVolleyball = fromItem.FeatureVolleyball;
66         }

```



```

67
68 /*-----
69 * Type:      Constructor
70 * Purpose:   Constructor that will fill all the properties except ItemID and ItemType.
71 * Input:    string name, contains the desired Name for the object.
72 * Input:    string type, contains the desired Type for the object.
73 * Input:    string streetAddress, contains the desired StreetAddress for the object.
74 * Input:    string city, contains the desired City for the object.
75 * Input:    string state, contains the desired State for the object.
76 * Input:    string zip, contains the desired Zip for the object.
77 * Input:    string latitude, contains the desired Latitude for the object.
78 * Input:    string longitude, contains the desired Longitude for the object.
79 * Input:    string phone, contains the desired Phone for the object.
80 * Input:    int featureBaseball, contains the desired FeatureBaseball for the object.
81 * Input:    float featureBasketball, contains the desired FeatureBasketball for the object.
82 * Input:    float featureGolf, contains the desired FeatureGolf for the object.
83 * Input:    int featureLargeMPField, contains the desired FeatureLargeMPField for the
84 *           object.
85 * Input:    int featureTennis, contains the desired FeatureTennis for the object.
86 * Input:    int featureVolleyball, contains the desired FeatureVolleyball for the object.
87 * Output:   Nothing.
88 -----*/
89 public Park(string name, string type, string streetAddress, string city, string state,
90            string zip, string latitude, string longitude, string phone, int featureBaseball,
91            float featureBasketball, float featureGolf, int featureLargeMPField,
92            int featureTennis, int featureVolleyball)
93     : base(name, type, streetAddress, city, state, zip, latitude, longitude, phone)
94 {
95     FeatureBaseball = featureBaseball;
96     FeatureBasketball = featureBasketball;
97     FeatureGolf = featureGolf;
98     FeatureLargeMPField = featureLargeMPField;
99     FeatureTennis = featureTennis;
100    FeatureVolleyball = featureVolleyball;
101 }
102
103 /*-----
104 * Type:      Auto-implemented Properties
105 -----*/
106 public override int ItemID { get; set; }           // Item ID
107
108 public override string Name { get; set; }          // Park Name
109 public override string Type { get; set; }          // Park Type
110 public override string StreetAddress { get; set; } // Street Address
111 public override string City { get; set; }          // City
112 public override string State { get; set; }         // State
113 public override string Zip { get; set; }           // Zip Code
114 public override string Latitude { get; set; }      // Latitude
115 public override string Longitude { get; set; }     // Longitude
116 public override string Phone { get; set; }         // Phone Number
117
118 /*-----
119 * Name:      ItemType
120 * Type:      Property
121 * Purpose:   Provides access to the itemType field.
122 -----*/
123 public override string ItemType
124 {
125     get
126     {
127         return itemType;
128     }
129     set
130     {
131         // Do nothing.
132     }

```

```

133     }
134
135     /*-----
136     * Name:     FeatureBaseball
137     * Type:     Property
138     * Purpose: Provides access to the featureBaseball field, and validation for the same.
139     -----*/
140     public int FeatureBaseball
141     {
142         get
143         {
144             return featureBaseball;
145         }
146         set
147         {
148             if (value >= 0)
149                 featureBaseball = value;
150         }
151     }
152
153     /*-----
154     * Name:     FeatureBasketball
155     * Type:     Property
156     * Purpose: Provides access to the featureBasketball field, and validation for the same.
157     -----*/
158     public float FeatureBasketball
159     {
160         get
161         {
162             return featureBasketball;
163         }
164         set
165         {
166             if (value >= 0 && value % 0.5 == 0)
167                 featureBasketball = value;
168         }
169     }
170
171     /*-----
172     * Name:     FeatureGolf
173     * Type:     Property
174     * Purpose: Provides access to the featureGolf field, and validation for the same.
175     -----*/
176     public float FeatureGolf
177     {
178         get
179         {
180             return featureGolf;
181         }
182         set
183         {
184             if (value >= 0 && value % 0.5 == 0)
185                 featureGolf = value;
186         }
187     }
188
189     /*-----
190     * Name:     FeatureLargeMPField
191     * Type:     Property
192     * Purpose: Provides access to the featureLargeMPField field, and validation for the same.
193     -----*/
194     public int FeatureLargeMPField
195     {
196         get
197         {
198             return featureLargeMPField;

```

```

199     }
200     set
201     {
202         if (value >= 0)
203             featureLargeMPField = value;
204     }
205 }
206
207 /*-----
208  * Name:     FeatureTennis
209  * Type:     Property
210  * Purpose:  Provides access to the featureTennis field, and validation for the same.
211  *-----*/
212 public int FeatureTennis
213 {
214     get
215     {
216         return featureTennis;
217     }
218     set
219     {
220         if (value >= 0)
221             featureTennis = value;
222     }
223 }
224
225 /*-----
226  * Name:     FeatureVolleyball
227  * Type:     Property
228  * Purpose:  Provides access to the featureVolleyball field, and validation for the same.
229  *-----*/
230 public int FeatureVolleyball
231 {
232     get
233     {
234         return featureVolleyball;
235     }
236     set
237     {
238         if (value >= 0)
239             featureVolleyball = value;
240     }
241 }
242
243 /*-----
244  * Name:     this[]
245  * Type:     Indexer
246  * Purpose:  Provides easy access to the properties of the class.
247  * Input:    FieldMenuHelper fiendNum, represents the desired property.
248  * Output:   object, contains whichever property was desired, or 0 if the property was not
249  *           found.
250  *-----*/
251 public override object this[FieldMenuHelper fieldNum]
252 {
253     get
254     {
255         switch (fieldNum)
256         {
257             case FieldMenuHelper.FeatureBaseball:
258                 return FeatureBaseball;
259             case FieldMenuHelper.FeatureBasketball:
260                 return FeatureBasketball;
261             case FieldMenuHelper.FeatureGolf:
262                 return FeatureGolf;
263             case FieldMenuHelper.FeatureLargeMPField:
264                 return FeatureLargeMPField;

```

```

265         case FieldMenuHelper.FeatureTennis:
266             return FeatureTennis;
267         case FieldMenuHelper.FeatureVolleyball:
268             return FeatureVolleyball;
269         default:
270             return base[fieldNum];
271     }
272 }
273 }
274
275 /*-----
276 * Name: ToStringCSV
277 * Type: Method
278 * Purpose: Serializes the data contained in the object into a comma-separated value string.
279 * Input: Nothing.
280 * Output: string, representing the data of this object as serialized to a CSV string.
281 -----*/
282 public override string ToStringCSV()
283 {
284     char separator = ',';
285     return base.ToStringCSV() + separator + FeatureBaseball + separator +
286         FeatureBasketball + separator + FeatureGolf + separator + FeatureLargeMPField +
287         separator + FeatureTennis + separator + FeatureVolleyball;
288 }
289
290 /*-----
291 * Name: ToString
292 * Type: Method
293 * Purpose: Override of ToString() method. Formats the data contained in this object so it
294 * looks pretty.
295 * Input: Nothing.
296 * Output: string, containing serialized object data.
297 -----*/
298 public override string ToString()
299 {
300     // Returns a the base a string formatted as follows:
301     // Item ID (Item Type): <ItemID> (<ItemType>)
302     // Park Name (Type): <Name> (<Type>)
303     // Address: <StreetAddress>, <City>, <State> <Zip>
304     // GPS Coordinates: (<Latitude>, <Longitude>)
305     // Phone Number: <Phone>
306     // Baseball Diamonds: <FeatureBaseball>
307     // Basketball Courts: <FeatureBasketball>
308     // Golf Courses: <FeatureGolf>
309     // Large MP Fields: <FeatureLargeMPField>
310     // Tennis Courts: <FeatureTennis>
311     // Volleyball Courts: <FeatureVolleyball>
312     return string.Format(
313         " Item ID (Item Type): {0} ({1})\n" +
314         " Park Name (Type): {2} ({3})\n" +
315         " Address: {4}, {5}, {6} {7}\n" +
316         " GPS Coordinates: ({8}, {9})\n" +
317         " Phone Number: {10}\n" +
318         " Baseball Diamonds: {11}\n" +
319         " Basketball Courts: {12}\n" +
320         " Golf Courses: {13}\n" +
321         " Large MP Fields: {14}\n" +
322         " Tennis Courts: {15}\n" +
323         " Volleyball Courts: {16}",
324         ItemID, ItemType,
325         Name, Type,
326         StreetAddress, City, State, Zip,
327         Latitude, Longitude,
328         Phone,
329         FeatureBaseball,
330         FeatureBasketball,

```

```
331         FeatureGolf,  
332         FeatureLargeMPField,  
333         FeatureTennis,  
334         FeatureVolleyball);  
335     }  
336 }  
337 }  
338
```

```

1  /*-----
2  * Author:      Dan Cassidy
3  * Date:        2015-06-17
4  * Assignment:  cView-P3
5  * Source File: PublicFacility.cs
6  * Language:    C#
7  * Course:      CSCI-C 490, C# Programming, MoWe 08:00
8  * Purpose:     Contains the PublicFacility class, derived from the Item abstract class, and
9  *              supporting methods.
10 -----*/
11
12 using System;
13 using System.Collections.Generic;
14 using System.Linq;
15 using System.Text;
16 using System.Threading.Tasks;
17
18 namespace Ph3
19 {
20     public class PublicFacility : Item
21     {
22         /*-----
23         * Type:      Helper Constants
24         * Purpose:
25         -----*/
26         public new const FieldMenuHelper FieldMin = FieldMenuHelper.Name;
27         public new const FieldMenuHelper FieldMax = FieldMenuHelper.Back;
28         public const int FieldOffset = 0;
29
30         /*-----
31         * Type:      Private Fields
32         -----*/
33         private string itemType = "publicfacility";
34
35         /*-----
36         * Type:      Constructor
37         * Purpose:    Basic no-parameter constructor.
38         * Input:      Nothing.
39         -----*/
40         public PublicFacility()
41         {
42             // Nothing else to do.
43         }
44
45         /*-----
46         * Type:      Constructor
47         * Purpose:    Copy constructor.
48         * Input:      PublicFacility fromItem, reference to the other PublicFacility from which fields
49         *              should be copied.
50         -----*/
51         public PublicFacility(PublicFacility fromItem)
52             : base(fromItem)
53         {
54             itemType = fromItem.itemType;
55         }
56
57         /*-----
58         * Type:      Constructor
59         * Purpose:    Constructor that will fill all the properties except ItemID and ItemType.
60         * Input:      string name, contains the desired Name for the object.
61         * Input:      string type, contains the desired Type for the object.
62         * Input:      string streetAddress, contains the desired StreetAddress for the object.
63         * Input:      string city, contains the desired City for the object.
64         * Input:      string state, contains the desired State for the object.
65         * Input:      string zip, contains the desired Zip for the object.
66         * Input:      string latitude, contains the desired Latitude for the object.

```

```

67      * Input:   string longitude, contains the desired Longitude for the object.
68      * Input:   string phone, contains the desired Phone for the object.
69      * Output:  Nothing.
70      -----*/
71      public PublicFacility(string name, string type, string streetAddress, string city, string state
72      ,
73      string zip, string latitude, string longitude, string phone)
74      : base(name, type, streetAddress, city, state, zip, latitude, longitude, phone)
75      {
76          // Nothing else to do.
77      }
78      /*-----
79      * Type:      Auto-implemented Properties
80      -----*/
81      public override int ItemID { get; set; }           // Item ID
82
83      public override string Name { get; set; }          // Facility Name
84      public override string Type { get; set; }          // Facility Type
85      public override string StreetAddress { get; set; } // Street Address
86      public override string City { get; set; }          // City
87      public override string State { get; set; }         // State
88      public override string Zip { get; set; }           // Zip Code
89      public override string Latitude { get; set; }      // Latitude
90      public override string Longitude { get; set; }     // Longitude
91      public override string Phone { get; set; }         // Phone Number
92
93      /*-----
94      * Name:      ItemType
95      * Type:      Property
96      * Purpose:   Provides access to the itemType field.
97      -----*/
98      public override string ItemType
99      {
100         get
101         {
102             return itemType;
103         }
104         set
105         {
106             // Do nothing.
107         }
108     }
109
110     /*-----
111     * Name:      ToString
112     * Type:      Method
113     * Purpose:   Override of ToString() method. Formats the data contained in this object so it
114     *             looks pretty.
115     * Input:     Nothing.
116     * Output:    string, containing serialized object data.
117     -----*/
118     public override string ToString()
119     {
120         // Returns a the base a string formatted as follows:
121         // Item ID (Item Type): <ItemID> (<ItemType>)
122         // Facility Name (Type): <Name> (<Type>)
123         // Address: <StreetAddress>, <City>, <State> <Zip>
124         // GPS Coordinates: (<Latitude>, <Longitude>)
125         // Phone Number: <Phone>
126         return string.Format(
127             " Item ID (Item Type): {0} ({1})\n" +
128             "Facility Name (Type): {2} ({3})\n" +
129             " Address: {4}, {5}, {6} {7}\n" +
130             " GPS Coordinates: ({8}, {9})\n" +
131             " Phone Number: {10}",

```

```
132         ItemID, ItemType,
133         Name, Type,
134         StreetAddress, City, State, Zip,
135         Latitude, Longitude,
136         Phone);
137     }
138
139 }
140 }
141
```



```

1  /*-----
2  * Author:      Dan Cassidy
3  * Date:        2015-06-17
4  * Assignment:  cView-P3
5  * Source File: SimpleConvert.cs
6  * Language:    C#
7  * Course:      CSCI-C 490, C# Programming, MoWe 08:00
8  * Purpose:     Provides some simplified variants of some Convert methods.
9  *-----*/
10
11 using System;
12 using System.Collections.Generic;
13 using System.Linq;
14 using System.Text;
15 using System.Threading.Tasks;
16
17 namespace Ph3
18 {
19     public static class SimpleConvert
20     {
21         /*-----
22         * Name:      ToDateTime
23         * Type:      Method
24         * Purpose: Attempts to convert the given parameter, but returns the default object value if
25         *           it fails for any reason.
26         * Input:    string value, containing the value on which conversion will be attempted.
27         * Output:   DateTime object representing either the converted value or the default DateTime.
28         *-----*/
29         public static DateTime ToDateTime(string value)
30         {
31             try
32             {
33                 return Convert.ToDateTime(value);
34             }
35             catch
36             {
37                 return default(DateTime);
38             }
39         }
40
41         /*-----
42         * Name:      ToInt32
43         * Type:      Method
44         * Purpose: Attempts to convert the given parameter, but returns the default object value if
45         *           it fails for any reason.
46         * Input:    string value, containing the value on which conversion will be attempted.
47         * Output:   int object representing either the converted value or the default int.
48         *-----*/
49         public static int ToInt32(string value)
50         {
51             try
52             {
53                 return Convert.ToInt32(value);
54             }
55             catch
56             {
57                 return default(int);
58             }
59         }
60
61         /*-----
62         * Name:      ToSingle
63         * Type:      Method
64         * Purpose: Attempts to convert the given parameter, but returns the default object value if
65         *           it fails for any reason.
66         * Input:    string value, containing the value on which conversion will be attempted.

```

```
67      * Output: float object representing either the converted value or the default float.
68      -----*/
69      public static float ToSingle(string value)
70      {
71          try
72          {
73              return Convert.ToSingle(value);
74          }
75          catch
76          {
77              return default(float);
78          }
79      }
80  }
81 }
82
```

Display all items on an empty dataset

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
| Main Interactive Menu |
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 6

| Display All Items |
No items to display.

| Main Interactive Menu |
```

Add new item 1 of 3 (business)

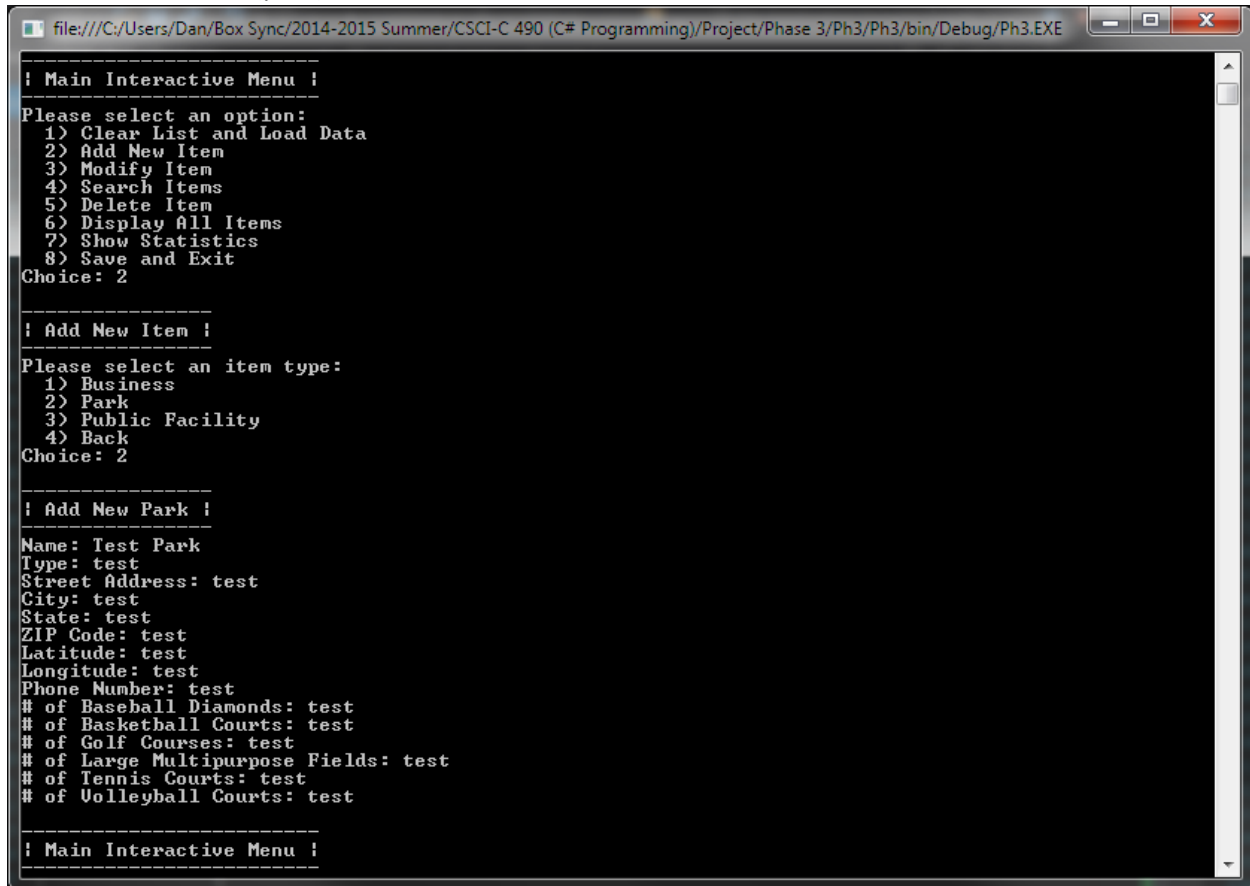
```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
| Main Interactive Menu |
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 2

| Add New Item |
Please select an item type:
1> Business
2> Park
3> Public Facility
4> Back
Choice: 1

| Add New Business |
Name: Test Business
Type: test
Street Address: test
City: test
State: test
ZIP Code: test
Latitude: test
Longitude: test
Phone Number: test
Business License Fiscal Year: test
Business License Number: test
Business License Issued Date: test
Business License Expiration Date: test
Business License Status: test
Council District: test

| Main Interactive Menu |
```

Add new item 2 of 3 (park)



```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE

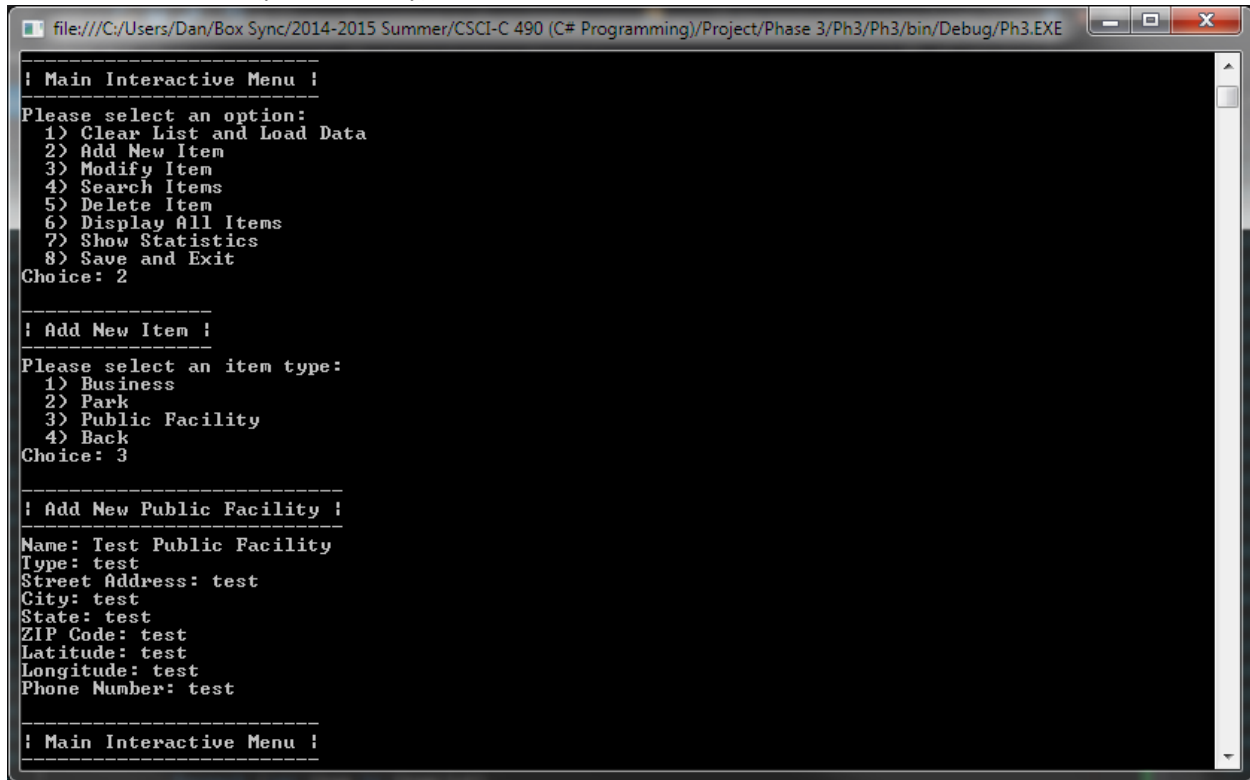
! Main Interactive Menu !
-----
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 2

! Add New Item !
-----
Please select an item type:
1> Business
2> Park
3> Public Facility
4> Back
Choice: 2

! Add New Park !
-----
Name: Test Park
Type: test
Street Address: test
City: test
State: test
ZIP Code: test
Latitude: test
Longitude: test
Phone Number: test
# of Baseball Diamonds: test
# of Basketball Courts: test
# of Golf Courses: test
# of Large Multipurpose Fields: test
# of Tennis Courts: test
# of Volleyball Courts: test

! Main Interactive Menu !
-----
```

Add new item 3 of 3 (public facility)



```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE

! Main Interactive Menu !
-----
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 2

! Add New Item !
-----
Please select an item type:
1> Business
2> Park
3> Public Facility
4> Back
Choice: 3

! Add New Public Facility !
-----
Name: Test Public Facility
Type: test
Street Address: test
City: test
State: test
ZIP Code: test
Latitude: test
Longitude: test
Phone Number: test

! Main Interactive Menu !
-----
```

Display all items from a small dataset

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
! Main Interactive Menu !
-----
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 6
-----
! Display All Items !
-----
Item ID <Item Type>: 1 <business>
Business Name <Type>: Test Business <test>
Address: test, test, test test
GPS Coordinates: <test, test>
Phone Number: test
License Number: 0-0
Valid: From 0001-01-01 to 0001-01-01
Status: test
Council District: test

Item ID <Item Type>: 2 <park>
Park Name <Type>: Test Park <test>
Address: test, test, test test
GPS Coordinates: <test, test>
Phone Number: test
Baseball Diamonds: 0
Basketball Courts: 0
Golf Courses: 0
Large MP Fields: 0
Tennis Courts: 0
Volleyball Courts: 0

Item ID <Item Type>: 3 <publicfacility>
Facility Name <Type>: Test Public Facility <test>
Address: test, test, test test
GPS Coordinates: <test, test>
Phone Number: test
-----
! Main Interactive Menu !
-----
```

## Modify 1 of 2

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
! Main Interactive Menu !
-----
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 3

! Modify Item -- Existing Items !
-----
Item ID: 1
Item Type: business
Name: Test Business

Item ID: 2
Item Type: park
Name: Test Park

Item ID: 3
Item Type: publicfacility
Name: Test Public Facility
Select item ID (<0 to cancel>): 1

! Modify Item -- Chosen Item !
-----
Item ID <Item Type>: 1 <business>
Business Name <Type>: Test Business <test>
Address: test, test, test test
GPS Coordinates: <test, test>
Phone Number: test
License Number: 0-0
Valid: From 0001-01-01 to 0001-01-01
Status: test
Council District: test

Please select the field you would like to work with:
1> Name
2> Type
3> Street Address
4> City
5> State
6> ZIP Code
7> Latitude
8> Longitude
9> Phone Number
10> Business License Fiscal Year
11> Business License Number
12> Business License Issued Date
13> Business License Expiration Date
14> Business License Status
15> Council District
16> Back
Choice: 10

Current Business License Fiscal Year: 0
New Business License Fiscal Year: 2001

! Modify Item -- Chosen Item !
-----
```

Modify 2 of 2

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
! Modify Item -- Chosen Item !
-----
Item ID <Item Type>: 1 <business>
Business Name <Type>: Test Business <test>
      Address: test, test, test test
      GPS Coordinates: <test, test>
      Phone Number: test
      License Number: 2001-0
      Valid: From 0001-01-01 to 0001-01-01
      Status: test
      Council District: test

Please select the field you would like to work with:
1> Name
2> Type
3> Street Address
4> City
5> State
6> ZIP Code
7> Latitude
8> Longitude
9> Phone Number
10> Business License Fiscal Year
11> Business License Number
12> Business License Issued Date
13> Business License Expiration Date
14> Business License Status
15> Council District
16> Back
Choice: 16

-----
! Main Interactive Menu !
-----
```



## Clear and Load

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
! Main Interactive Menu !
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 1

! Load Files !
Please select an item type:
1> Business
2> Park
3> Public Facility
4> Back
Choice: 1
Enter a filename to load: businesses.base.csv
14057 items loaded.
Please select an item type:
1> Business
2> Park
3> Public Facility
4> Back
Choice: 2
Enter a filename to load: parks.base.csv
62 items loaded.
Please select an item type:
1> Business
2> Park
3> Public Facility
4> Back
Choice: 3
Enter a filename to load: publicfacilities.base.csv
51 items loaded.
Please select an item type:
1> Business
2> Park
3> Public Facility
4> Back
Choice: 4

! Main Interactive Menu !
```

## Statistics

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE

! Main Interactive Menu !
-----
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 7

-----
! Statistics !
-----
50 unique types of items found.

adult business: 54
alarm agent: 445
arborist/tree service: 409
automotive repair & svc: 946
block park: 16
busker or sidewalk performer: 13
cemetery: 2
charitable solicitations: 133
community park: 13
donation boxes and containers: 122
fire station: 35
food vending machines: 4
food vending vehicles: 31
golf course: 4
hotel/motel: 195
itinerant restaurant: 20
lawn park 10 or more spaces: 595
lawn park under 10 spaces: 576
library: 13
massage establishment: 102
massage therapy: 205
mechanical license: 10
memorial: 1
neighborhood park: 24
open air vendors private: 98
open air vendors public: 66
outdoor movie theatre: 1
peddler & canvasser: 573
performing animal exhibitions: 1
pet shops: 39
police station: 3
pool halls: 8
precious metal dealers: 96
public parking facility: 149
restaurants a-m: 2927
restaurants n-z: 1888
rubbish & garbage removal: 179
scrap metal /junk dealers: 168
second hand dealers: 292
self service laundry: 99
special: 1
tattoo and body piercing establishments: 3
tattoo and piercing artist: 14
taxi company: 114
taxi driver: 1720
taxi vehicle: 1599
transient merchant: 85
transient merchant xmas tree: 3
vehicle removal: 75
zoo: 1

-----
! Main Interactive Menu !
-----
```

Display All w/ pagination

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE

! Main Interactive Menu !
Please select an option:
1) Clear List and Load Data
2) Add New Item
3) Modify Item
4) Search Items
5) Delete Item
6) Display All Items
7) Show Statistics
8) Save and Exit
Choice: 6

! Display All Items !

Item ID <Item Type>: 1 <business>
Business Name <Type>: ARBY'S ROAST BEEF #677 <RESTAURANTS A-M>
Address: 1807 LINCOLN WAY E, SOUTH BEND, IN 46613-3422
GPS Coordinates: <41.657927716000074, -86.22006660399995>
Phone Number: <574> 299-6487
License Number: 2000-955
Valid: From 2006-04-18 to 2007-02-28
Status: Renewed
Council District:

Item ID <Item Type>: 2 <business>
Business Name <Type>: ECONO LODGE <HOTEL/MOTEL>
Address: 3233 LINCOLN WAY W, SOUTH BEND, IN 46628-1450
GPS Coordinates: <41.691619321000076, -86.29479799599994>
Phone Number: <574> 232-9019
License Number: 2004-1
Valid: From 2004-01-20 to 2005-01-31
Status: Active and Licensed
Council District:

Item ID <Item Type>: 3 <business>
Business Name <Type>: ABC CAB <DON'T USE> <TAXI COMPANY>
Address: 1733 S MICHIGAN ST, SOUTH BEND, IN 46613
GPS Coordinates: <41.65644480900005, -86.24989745499994>
Phone Number: <574> 233-4000
License Number: 2004-2
Valid: From 2004-04-22 to 2005-06-01
Status: Renewed
Council District: COUNCIL DISTRICT 3

Enter for next item. Space for next page. Ctrl+Enter for all. Esc to abort.

Item ID <Item Type>: 4 <business>
Business Name <Type>: BFI WASTE SERVICES <RUBBISH & GARBAGE REMOVAL>
Address: 1 OUT OF AREA AVE, SOUTH BEND, IN 46601
GPS Coordinates: <41.67277586800003, -86.25337886399996>
Phone Number: <574> 522-1331
License Number: 2004-4
Valid: From 2004-10-22 to 2005-04-30
Status: Renewed
Council District:

Item ID <Item Type>: 5 <business>
Business Name <Type>: PETE BUCHANAN <RUBBISH & GARBAGE REMOVAL>
Address: 1210 BISSELL ST, SOUTH BEND, IN 46617
GPS Coordinates: <41.68436336600007, -86.23241099899997>
Phone Number: <574> 282-1588
License Number: 2004-5
Valid: From 2004-05-04 to 2005-04-30
Status: Inactive and no longer licensed
Council District: COUNCIL DISTRICT 4

Item ID <Item Type>: 6 <business>
Business Name <Type>: CLARK TRASH REMOVAL <RUBBISH & GARBAGE REMOVAL>
Address: 1146 THOMAS ST, , 46625
GPS Coordinates: <, >
Phone Number: <574> 287-8028
License Number: 2004-6
Valid: From 2004-05-06 to 2005-04-30
Status: Renewed
Council District: COUNCIL DISTRICT 2

Enter for next item. Space for next page. Ctrl+Enter for all. Esc to abort.

! Main Interactive Menu !
```

## Searching a larger dataset 1 of 2

```

file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
! Main Interactive Menu !
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 4

! Search Items !
Please select an item type:
1> Business
2> Park
3> Public Facility
4> Back
Choice: 1

! Search Businesses !
Please select the field you would like to work with:
1> Name
2> Type
3> Street Address
4> City
5> State
6> ZIP Code
7> Latitude
8> Longitude
9> Phone Number
10> Business License Fiscal Year
11> Business License Number
12> Business License Issued Date
13> Business License Expiration Date
14> Business License Status
15> Council District
16> Back
Choice: 10

! Search Items -- Business License Fiscal Year !
What kind of comparator do you wish to use?
! - contains (default)      >= - greater than or equal to
?! - does not contain      <= - less than or equal to
= - equal                  > - greater than
!= - not equal             < - less than
Choice: <

Enter your search text: 2005

! Search Results !
166 items found.

Item ID <Item Type>: 1 <business>
Business Name <Type>: ARBY'S ROAST BEEF #677 <RESTAURANTS A-M>
Address: 1807 LINCOLN WAY E. SOUTH BEND, IN 46613-3422
GPS Coordinates: <41.657927716000074, -86.22006660399995>
Phone Number: <574> 299-6487
License Number: 2000-955
Valid: From 2006-04-18 to 2007-02-28
Status: Renewed
Council District:

Item ID <Item Type>: 2 <business>
Business Name <Type>: ECONO LODGE <HOTEL/MOTEL>
Address: 3233 LINCOLN WAY W. SOUTH BEND, IN 46628-1450
GPS Coordinates: <41.691619321000076, -86.29479799599994>
Phone Number: <574> 232-9019
License Number: 2004-1
Valid: From 2004-01-20 to 2005-01-31
Status: Active and Licensed
Council District:

```

Searching a larger dataset 2 of 2

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE

Item ID <Item Type>: 3 <business>
Business Name <Type>: ABC CAB <DON'T USE> <TAXI COMPANY>
Address: 1733 S MICHIGAN ST, SOUTH BEND, IN 46613
GPS Coordinates: (41.65644480900005, -86.24989745499994)
Phone Number: (574) 233-4000
License Number: 2004-2
Valid: From 2004-04-22 to 2005-06-01
Status: Renewed
Council District: COUNCIL DISTRICT 3

Enter for next item. Space for next page. Ctrl+Enter for all. Esc to abort.

Item ID <Item Type>: 4 <business>
Business Name <Type>: BFI WASTE SERVICES <RUBBISH & GARBAGE REMOVAL>
Address: 1 OUT OF AREA AVE, SOUTH BEND, IN 46601
GPS Coordinates: (41.67277586800003, -86.25337886399996)
Phone Number: (574) 522-1331
License Number: 2004-4
Valid: From 2004-10-22 to 2005-04-30
Status: Renewed
Council District:

Item ID <Item Type>: 5 <business>
Business Name <Type>: PETE BUCHANAN <RUBBISH & GARBAGE REMOVAL>
Address: 1210 BISSELL ST, SOUTH BEND, IN 46617
GPS Coordinates: (41.68436336600007, -86.23241099899997)
Phone Number: (574) 282-1588
License Number: 2004-5
Valid: From 2004-05-04 to 2005-04-30
Status: Inactive and no longer licensed
Council District: COUNCIL DISTRICT 4

Item ID <Item Type>: 6 <business>
Business Name <Type>: CLARK TRASH REMOVAL <RUBBISH & GARBAGE REMOVAL>
Address: 1146 THOMAS ST, , 46625
GPS Coordinates: (, )
Phone Number: (574) 287-8028
License Number: 2004-6
Valid: From 2004-05-06 to 2005-04-30
Status: Renewed
Council District: COUNCIL DISTRICT 2

Enter for next item. Space for next page. Ctrl+Enter for all. Esc to abort.

Item ID <Item Type>: 7 <business>
Business Name <Type>: AGAPE CAB COMPANY <TAXI COMPANY>
Address: 1843 COMMERCE DR, SOUTH BEND, IN 46628
GPS Coordinates: (41.699999830000024, -86.30750750999994)
Phone Number: (574) 514-2993
License Number: 2004-7
Valid: From 2004-03-18 to 2005-06-01
Status: Renewed
Council District: COUNCIL DISTRICT 1

Enter for next item. Space for next page. Ctrl+Enter for all. Esc to abort.

! Search Businesses !

Please select the field you would like to work with:
1) Name
2) Type
3) Street Address
4) City
5) State
6) ZIP Code
7) Latitude
8) Longitude
9) Phone Number
10) Business License Fiscal Year
11) Business License Number
12) Business License Issued Date
13) Business License Expiration Date
14) Business License Status
15) Council District
16) Back
Choice: 16

! Search Items !

Please select an item type:
1) Business
2) Park
3) Public Facility
4) Back
Choice: 4

! Main Interactive Menu !
```

## Delete item

```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
! Main Interactive Menu !
-----
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 5

! Delete Item -- Existing Items !
-----
Item ID: 1
Item Type: business
Name: ARBY'S ROAST BEEF #677

Item ID: 2
Item Type: business
Name: ECONO LODGE

Item ID: 3
Item Type: business
Name: ABC CAB <DON'T USE>

Item ID: 4
Item Type: business
Name: BFI WASTE SERVICES

Item ID: 5
Item Type: business
Name: PETE BUCHANAN

Item ID: 6
Item Type: business
Name: CLARK TRASH REMOVAL

Item ID: 7
Item Type: business
Name: AGAPE CAB COMPANY

Enter for next item. Space for next page. Ctrl+Enter for all. Esc to abort.
Select item ID (<0 to cancel): 1

! Delete Item -- Results !
-----
Item ID 1 has been deleted.

Item ID: 2
Item Type: business
Name: ECONO LODGE

Item ID: 3
Item Type: business
Name: ABC CAB <DON'T USE>

Item ID: 4
Item Type: business
Name: BFI WASTE SERVICES

Item ID: 5
Item Type: business
Name: PETE BUCHANAN

Item ID: 6
Item Type: business
Name: CLARK TRASH REMOVAL

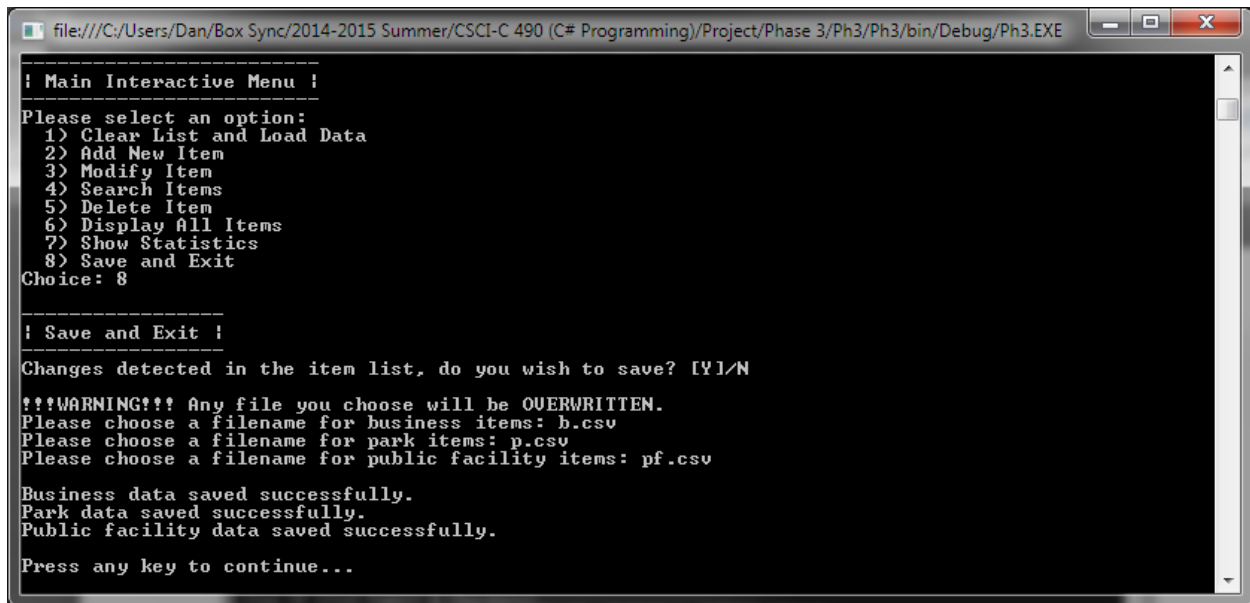
Item ID: 7
Item Type: business
Name: AGAPE CAB COMPANY

Item ID: 8
Item Type: business
Name: JAMISON INN

Enter for next item. Space for next page. Ctrl+Enter for all. Esc to abort.

! Main Interactive Menu !
-----
```

Save on exit



```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE

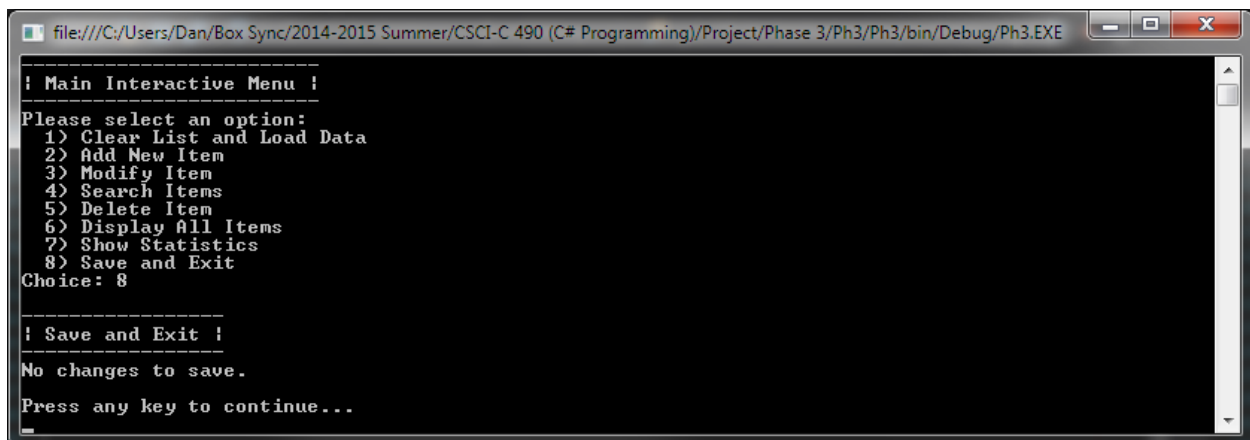
! Main Interactive Menu !
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 8

! Save and Exit !
Changes detected in the item list, do you wish to save? [Y]/N
???WARNING!!! Any file you choose will be OVERWRITTEN.
Please choose a filename for business items: b.csv
Please choose a filename for park items: p.csv
Please choose a filename for public facility items: pf.csv

Business data saved successfully.
Park data saved successfully.
Public facility data saved successfully.

Press any key to continue...
```

Doesn't save if not needed.



```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE

! Main Interactive Menu !
Please select an option:
1> Clear List and Load Data
2> Add New Item
3> Modify Item
4> Search Items
5> Delete Item
6> Display All Items
7> Show Statistics
8> Save and Exit
Choice: 8

! Save and Exit !
No changes to save.

Press any key to continue...
```