

```

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
```