```
/*--------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy and Dr. Raman Adaikkalavan
 * Date:        2015-06-17
 * Assignment:  cView-P3
 * Source File: ItemDB.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Encapsulates a List-based collection of Item objects and contains related methods
 *              and properties.
--------------------------------------------------------------------------------------------*/

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

namespace Ph3
{
    public class ItemDB : IEnumerable
    {
        /*--------------------------------------------------------------------------------------
         * Name:    KeyStart
         * Type:    Constant
         * Purpose: Contains the default value for currentItemKey.
        --------------------------------------------------------------------------------------*/
        private const int KeyStart = 1;

        /*--------------------------------------------------------------------------------------
         * Name:    currentItemKey
         * Type:    Field
         * Purpose: Implements a counter for the ID number for Item objects. This is due to the fact
         *          that itemList.Count becomes unreliable if objects are removed from the list.
        --------------------------------------------------------------------------------------*/
        private static int currentItemKey = KeyStart;

        /*--------------------------------------------------------------------------------------
         * Name:    itemList
         * Type:    Field
         * Purpose: List of Item objects.
        --------------------------------------------------------------------------------------*/
        private List<Item> itemList = new List<Item>();

        /*--------------------------------------------------------------------------------------
         * Name:    Count
         * Type:    Property
         * Purpose: Enable access to the Count property of itemList.
        --------------------------------------------------------------------------------------*/
        public int Count { get { return itemList.Count; } }

        /*--------------------------------------------------------------------------------------
         * Name:    IsChanged
         * Type:    Property
         * Purpose: Flag that determines whether the itemDB has been modified (true) or not (false).
        --------------------------------------------------------------------------------------*/
        public bool IsChanged { get; private set; }

        /*--------------------------------------------------------------------------------------
         * Name:    Add
         * Type:    Method
         * Purpose: Add the specified item object to the ItemDB object.
         * Input:   Item item, specifies the item to be added to the ItemDB object.
         * Output:  Nothing.
        --------------------------------------------------------------------------------------*/
        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 
315                 case "=":
316                     if (baseObject is DateTime)
317                         return new ItemDB() { itemList = typeLimitedList.
318                             Where(i => (DateTime)i[field] == SimpleConvert.ToDateTime(toSearchFor)).
319                             ToList() };
320                     else if (baseObject is float)
321                         return new ItemDB() { itemList = typeLimitedList.
322                             Where(i => (float)i[field] == SimpleConvert.ToSingle(toSearchFor)).
323                             ToList() };
324                     else if (baseObject is int)
325                         return new ItemDB() { itemList = typeLimitedList.
326                             Where(i => (int)i[field] == SimpleConvert.ToInt32(toSearchFor)).
327                             ToList() };
328                     else
329                         return new ItemDB() { itemList = typeLimitedList.
330                             Where(i => (string)i[field] == toSearchFor).
```

```
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
```