```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy and Dr. Raman Adaikkalavan
 * Date:        2015-06-17
 * Assignment:  cView-P3
 * Source File: ItemDBTest.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Project:     The overall goal of this project is to capitalize on the fact that government, from
 *              local to national, has made some of its data open by developing a way to explore
 *              this data and present it to a user in a meaningful fashion. This phase of the
 *              project is meant to explore data from any combination of the Business dataset
 *              (https://data.southbendin.gov/d/imxu-7m5i), the Parks and Features dataset
 *              (https://data.southbendin.gov/d/yf5x-7tkb), and the Public Facility dataset
 *              (https://data.southbendin.gov/d/jeef-dsq9).
 * Purpose:     Small wrapper program for demonstrating the ItemDBInteractive class.
-----------------------------------------------------------------------------------------*/

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

namespace Ph3
{
    public class ItemDBTest
    {
        /*-----------------------------------------------------------------------------------
         * Name:    Main
         * Type:    Method
         * Purpose: Serves as the entry point to the program.
         * Input:   (Ignored) string[] args, represents any command line arguments.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------*/
        static void Main(string[] args)
        {
            // Declare a new ItemDBInteractive object and interactively manipulate said object.
            ItemDBInteractive itemDBTest = new ItemDBInteractive();
            itemDBTest.InteractiveManipulation();
        }
    }
}

```

```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-17
 * Assignment:  cView-P3
 * Source File: ItemDBInteractive.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Provides interactive management of an ItemDB object.
-----------------------------------------------------------------------------------------*/

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

namespace Ph3
{
    public class ItemDBInteractive
    {
        /*-----------------------------------------------------------------------------------
         * Type:    Helper Constants
         * Purpose: Menu validation.
        -----------------------------------------------------------------------------------*/
        private const MainMenu MainMenuMin = MainMenu.Load;
        private const MainMenu MainMenuMax = MainMenu.Exit;
        private const TypeMenu TypeMenuMin = TypeMenu.Business;
        private const TypeMenu TypeMenuMax = TypeMenu.Back;

        /*-----------------------------------------------------------------------------------
         * Name:    itemDB
         * Type:    Field
         * Purpose: ItemDB that this class works with.
        -----------------------------------------------------------------------------------*/
        private ItemDB itemDB = new ItemDB();

        /*-----------------------------------------------------------------------------------
         * Name:    MainMenu
         * Type:    Enum
         * Purpose: Enum for the main menu. Basic code idea from Stack Overflow.
         *          http://stackoverflow.com/a/15752719
        -----------------------------------------------------------------------------------*/
        private enum MainMenu
        {
            Load = 1,
            Add,
            Modify,
            Search,
            Delete,
            DisplayAll,
            Statistics,
            Exit
        }

        /*-----------------------------------------------------------------------------------
         * Name:    TypeMenu
         * Type:    Enum
         * Purpose: Enum for the type menu. Basic code idea from Stack Overflow.
         *          http://stackoverflow.com/a/15752719
        -----------------------------------------------------------------------------------*/
        private enum TypeMenu
        {
            Business = 1,
            Park,
```

```
            PublicFacility,
            Back
        }

        /*-------------------------------------------------------------------------------------
         * Name:    InteractiveManipulation
         * Type:    Method
         * Purpose: Entry point for interactive manipulation of ItemDB object.
         * Input:   Nothing.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        public void InteractiveManipulation()
        {
            //Loop the main menu until the user decides to exit.
            while (MainMenuAction(MainMenuDisplay()) != MainMenu.Exit) ;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    DataAdd
         * Type:    Method
         * Purpose: Interactively add an item based on the user's input.
         * Input:   Nothing.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        private void DataAdd()
        {
            // Declare a reference for parent class.
            Item itemToAdd;

            // Prompt the user to choose what type of item to add.
            Console.WriteLine("----------------");
            Console.WriteLine("| Add New Item |");
            Console.WriteLine("----------------");
            TypeMenu choice = TypeMenuDisplay();

            // Determine what the user wishes to do.
            switch (choice)
            {
                case TypeMenu.Business:
                    itemToAdd = new Business();
                    Console.WriteLine("--------------------");
                    Console.WriteLine("| Add New Business |");
                    Console.WriteLine("--------------------");
                    break;

                case TypeMenu.Park:
                    itemToAdd = new Park();
                    Console.WriteLine("----------------");
                    Console.WriteLine("| Add New Park |");
                    Console.WriteLine("----------------");
                    break;

                case TypeMenu.PublicFacility:
                    itemToAdd = new PublicFacility();
                    Console.WriteLine("---------------------------");
                    Console.WriteLine("| Add New Public Facility |");
                    Console.WriteLine("---------------------------");
                    break;

                case TypeMenu.Back:
                    // Nothing to do; user wants to go back.
                default:
                    // Catch-all.
                    return;
            }

```

```
            // Handle filling in the common fields
            Console.Write("Name: ");
            itemToAdd.Name = Console.ReadLine();

            Console.Write("Type: ");
            itemToAdd.Type = Console.ReadLine();

            Console.Write("Street Address: ");
            itemToAdd.StreetAddress = Console.ReadLine();

            Console.Write("City: ");
            itemToAdd.City = Console.ReadLine();

            Console.Write("State: ");
            itemToAdd.State = Console.ReadLine();

            Console.Write("ZIP Code: ");
            itemToAdd.Zip = Console.ReadLine();

            Console.Write("Latitude: ");
            itemToAdd.Latitude = Console.ReadLine();

            Console.Write("Longitude: ");
            itemToAdd.Longitude = Console.ReadLine();

            Console.Write("Phone Number: ");
            itemToAdd.Phone = Console.ReadLine();

            // Check whether the Item object is a Business object or Park object.
            if (itemToAdd is Business)
            {
                // Business object. Handle Business object-specific fields.
                Business businessToAdd = itemToAdd as Business;

                Console.Write("Business License Fiscal Year: ");
                businessToAdd.LicenseFiscalYear = SimpleConvert.ToInt32(Console.ReadLine());

                Console.Write("Business License Number: ");
                businessToAdd.LicenseNumber = SimpleConvert.ToInt32(Console.ReadLine());

                Console.Write("Business License Issued Date): ");
                businessToAdd.LicenseIssueDate = SimpleConvert.ToDateTime(Console.ReadLine());

                Console.Write("Business License Expiration Date: ");
                businessToAdd.LicenseExpirDate = SimpleConvert.ToDateTime(Console.ReadLine());

                Console.Write("Business License Status: ");
                businessToAdd.LicenseStatus = Console.ReadLine();

                Console.Write("Council District: ");
                businessToAdd.CouncilDistrict = Console.ReadLine();
            }
            else if (itemToAdd is Park)
            {
                // Park object. Handle Park object-specific fields.
                Park parkToAdd = itemToAdd as Park;

                Console.Write("# of Baseball Diamonds: ");
                parkToAdd.FeatureBaseball = SimpleConvert.ToInt32(Console.ReadLine());

                Console.Write("# of Basketball Courts: ");
                parkToAdd.FeatureBasketball = SimpleConvert.ToSingle(Console.ReadLine());

                Console.Write("# of Golf Courses: ");
                parkToAdd.FeatureGolf = SimpleConvert.ToSingle(Console.ReadLine());

```

```
                Console.Write("# of Large Multipurpose Fields: ");
                parkToAdd.FeatureLargeMPField = SimpleConvert.ToInt32(Console.ReadLine());

                Console.Write("# of Tennis Courts: ");
                parkToAdd.FeatureTennis = SimpleConvert.ToInt32(Console.ReadLine());

                Console.Write("# of Volleyball Courts: ");
                parkToAdd.FeatureVolleyball = SimpleConvert.ToInt32(Console.ReadLine());
            }

            // Extra line for formatting.
            Console.WriteLine();

            // Add the new item to the main data set.
            itemDB.Add(itemToAdd);
        }

        /*-----------------------------------------------------------------------------------------
         * Name:    DataDelete
         * Type:    Method
         * Purpose: Interactively deletes an object based upon user input.
         * Input:   Nothing.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        private void DataDelete()
        {
            // Display the user's choice.
            Console.WriteLine("---------------------------------");
            Console.WriteLine("| Delete Item -- Existing Items |");
            Console.WriteLine("---------------------------------");

            // Display a simple list of all the objects in the data set.
            itemDB.DisplayAll(true);

            // Get the user's choice of which object to delete.
            Console.Write("Select item ID (0 to cancel): ");
            int itemIDToDelete = SimpleConvert.ToInt32(Console.ReadLine());
            int indexToDelete = itemDB.GetItemIndex(itemIDToDelete);

            // Extra line for formatting.
            Console.WriteLine();

            // Display the results.
            Console.WriteLine("--------------------------");
            Console.WriteLine("| Delete Item -- Results |");
            Console.WriteLine("--------------------------");

            // Validate the user's choice.
            if (itemIDToDelete == 0)
            {
                // The user changed their mind.
                Console.WriteLine("Cancelled.\n");
                return;
            }
            else if (itemIDToDelete < 0 || indexToDelete < 0)
            {
                // The user input an invalid object index.
                Console.WriteLine("Invalid item.\n");
                return;
            }

            // Delete the object and display confirmation of its deletion.
            if (itemDB.Delete(itemIDToDelete))
                Console.WriteLine("Item ID {0} has been deleted.\n", itemIDToDelete);
            else
                Console.WriteLine("Error occured while attempting to delete item ID {0}.\n",
```

```
                    itemIDToDelete);

            // Display a simple list of the still existing items.
            itemDB.DisplayAll(true);
        }

        /*-------------------------------------------------------------------------------------
         * Name:    DataDisplayAll
         * Type:    Method
         * Purpose: Displayes a list of all items.
         * Input:   Nothing.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        private void DataDisplayAll()
        {
            // Display the user's choice.
            Console.WriteLine("---------------------");
            Console.WriteLine("| Display All Items |");
            Console.WriteLine("---------------------");

            // Display all the items.
            itemDB.DisplayAll();
        }

        /*-------------------------------------------------------------------------------------
         * Name:    DataLoad
         * Type:    Method
         * Purpose: Get the user's choice of CSV files to import.
         * Input:   Nothing.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        private void DataLoad()
        {
            TypeMenu typeChoice;

            itemDB.Reset();

            // Display the user's choice.
            Console.WriteLine("--------------");
            Console.WriteLine("| Load Files |");
            Console.WriteLine("--------------");

            // Read files for as long as the user wants.
            while ((typeChoice = TypeMenuDisplay()) != TypeMenu.Back)
            {
                Console.Write("Enter a filename to load: ");

                try
                {
                    int tempCount = itemDB.Count;
                    DataLoadProcessFile(Console.ReadLine(), typeChoice);
                    Console.WriteLine("{0} item{1} loaded.", itemDB.Count - tempCount,
                        (itemDB.Count - tempCount != 1) ? "s" : "");
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                }

                Console.WriteLine();
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    DataLoadProcessFile
         * Type:    Method
```

```
         * Purpose: Process the file specified and add the resulting Items to the ItemDB. Used some
         *          of the code from the book example Fig17_11 as a starting point.
         * Input:   string fileName, contains the filename to process.
         * Input:   TypeMenu itemType, contains the type of item to add.
         * Output:  Nothing.
        ----------------------------------------------------------------------------------------*/
        private void DataLoadProcessFile(string fileName, TypeMenu itemType)
        {
            using (StreamReader fileReader = new StreamReader(fileName))
            {
                string inputItem = fileReader.ReadLine();
                string[] inputFields;

                while (inputItem != null)
                {
                    Item toAdd = null;
                    inputFields = inputItem.Split(',');

                    // Process a line based on what type is being imported.
                    switch (itemType)
                    {
                        case TypeMenu.Business:
                            toAdd = new Business(inputFields[0], inputFields[1], inputFields[2],
                                inputFields[3], inputFields[4], inputFields[5], inputFields[6],
                                inputFields[7], inputFields[8],
                                SimpleConvert.ToInt32(inputFields[9].Split('-')[0]),
                                SimpleConvert.ToInt32(inputFields[9].Split('-')[1]),
                                SimpleConvert.ToDateTime(inputFields[10]),
                                SimpleConvert.ToDateTime(inputFields[11]), inputFields[12],
                                inputFields[13]);
                            break;

                        case TypeMenu.Park:
                            toAdd = new Park(inputFields[0], inputFields[1], inputFields[2],
                                inputFields[3], inputFields[4], inputFields[5], inputFields[6],
                                inputFields[7], inputFields[8],
                                SimpleConvert.ToInt32(inputFields[9]),
                                SimpleConvert.ToSingle(inputFields[10]),
                                SimpleConvert.ToSingle(inputFields[11]),
                                SimpleConvert.ToInt32(inputFields[12]),
                                SimpleConvert.ToInt32(inputFields[13]),
                                SimpleConvert.ToInt32(inputFields[14]));
                            break;

                        case TypeMenu.PublicFacility:
                            toAdd = new PublicFacility(inputFields[0], inputFields[1],
                                inputFields[2], inputFields[3], inputFields[4], inputFields[5],
                                inputFields[6], inputFields[7], inputFields[8]);
                            break;

                        default:
                            break;
                    }

                    if (toAdd != null)
                        itemDB.Add(toAdd);

                    inputItem = fileReader.ReadLine();
                }
            }
        }

        /*----------------------------------------------------------------------------------------
         * Name:    DataModify
         * Type:    Method
         * Purpose: Interactively modifies an object based on the user's input.
```

```
         * Input:   Nothing.
         * Output:  Nothing.
        ----------------------------------------------------------------------------------------*/
        private void DataModify()
        {
            //Display the user's choice.
            Console.WriteLine("--------------------------------");
            Console.WriteLine("| Modify Item -- Existing Items |");
            Console.WriteLine("--------------------------------");

            // Display a simple list of all the objects in the data set.
            itemDB.DisplayAll(true);

            // Get the user's choice of which object to delete.
            Console.Write("Select item ID (0 to cancel): ");
            int itemIDToModify = SimpleConvert.ToInt32(Console.ReadLine());
            int indexToModify = itemDB.GetItemIndex(itemIDToModify);

            // Extra line for formatting.
            Console.WriteLine();

            // Validate the user's choice.
            if (itemIDToModify == 0)
            {
                // The user changed their mind.
                Console.WriteLine("Cancelled.\n");
                return;
            }
            else if (itemIDToModify < 0 || indexToModify < 0)
            {
                // The user input an invalid object index.
                Console.WriteLine("Invalid item.\n");
                return;
            }

            // Store reference to item copy.
            Item itemToModify = itemDB.GetItem(itemIDToModify);

            do
            {
                // Display the chosen object.
                Console.WriteLine("------------------------------");
                Console.WriteLine("| Modify Item -- Chosen Item |");
                Console.WriteLine("------------------------------");
                Console.WriteLine("{0}\n", itemToModify);

                // Loop while the use has not chosen to go back.
            } while (DataModifyMenuAction(FieldMenuDisplay(itemToModify),
                itemToModify) != Item.FieldMenuHelper.Back);
        }

        /*----------------------------------------------------------------------------------------
         * Name:    DataModifyMenuAction
         * Type:    Method
         * Purpose: Acts on the user's choice made at the Modify Menu.
         * Input:   Item.FieldMenuHelper choice, represents the action specified.
         * Input:   Item itemToModify, is a copy of the object that will be modified.
         * Output:  Item.FieldMenuHelper, represents the action specified.
        ----------------------------------------------------------------------------------------*/
        private Item.FieldMenuHelper DataModifyMenuAction(Item.FieldMenuHelper choice,
            Item itemToModify)
        {
            // Handle the common fields of an Item object.
            switch (choice)
            {
                case Item.FieldMenuHelper.Name:
```

```
                    // Change the Name of the item.
                    Console.WriteLine("Current Name: {0}", itemToModify.Name);
                    Console.Write("New Name: ");
                    itemToModify.Name = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.Type:
                    // Change the Type of the item.
                    Console.WriteLine("Current Type: {0}", itemToModify.Type);
                    Console.Write("New Type: ");
                    itemToModify.Type = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.StreetAddress:
                    // Change the StreetAddress of the item.
                    Console.WriteLine("Current Street Address: {0}", itemToModify.StreetAddress);
                    Console.Write("New Street Address: ");
                    itemToModify.StreetAddress = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.City:
                    // Change the City of the item.
                    Console.WriteLine("Current City: {0}", itemToModify.City);
                    Console.Write("New City: ");
                    itemToModify.City = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.State:
                    // Change the State of the item.
                    Console.WriteLine("Current State: {0}", itemToModify.State);
                    Console.Write("New State: ");
                    itemToModify.State = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.Zip:
                    // Change the Zip of the item.
                    Console.WriteLine("Current ZIP Code: {0}", itemToModify.Zip);
                    Console.Write("New ZIP Code: ");
                    itemToModify.Zip = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.Latitude:
                    // Change the Latitude of the item.
                    Console.WriteLine("Current Latitude: {0}", itemToModify.Latitude);
                    Console.Write("New Latitude: ");
                    itemToModify.Latitude = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.Longitude:
                    // Change the Longitude of the item.
                    Console.WriteLine("Current Longitude: {0}", itemToModify.Longitude);
                    Console.Write("New Longitude: ");
                    itemToModify.Longitude = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.Phone:
                    // Change the Phone of the item.
                    Console.WriteLine("Current Phone Number: {0}", itemToModify.Phone);
                    Console.Write("New Phone Number: ");
                    itemToModify.Phone = Console.ReadLine();
                    break;

                case Item.FieldMenuHelper.Back:
                case Item.FieldMenuHelper.BackBusiness:
                case Item.FieldMenuHelper.BackPark:
                    // Nothing to do; the user wants to go back.
```

```
                    return Item.FieldMenuHelper.Back;

                default:
                    // Catch-all.
                    break;
            }

            // Check whether the Item object is a Business object or Park object.
            if (itemToModify is Business)
            {
                // Business object. Handle Business object-specific fields.
                Business businessToModify = itemToModify as Business;

                switch (choice)
                {
                    case Item.FieldMenuHelper.LicenseFiscalYear:
                        // Change the LicenseFiscalYear of the business.
                        Console.WriteLine("Current Business License Fiscal Year: {0}",
                            businessToModify.LicenseFiscalYear);
                        Console.Write("New Business License Fiscal Year: ");
                        businessToModify.LicenseFiscalYear =
                            SimpleConvert.ToInt32(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.LicenseNumber:
                        // Change the LicenseNumber of the business.
                        Console.WriteLine("Current Business License Number: {0}",
                            businessToModify.LicenseNumber);
                        Console.Write("New Business License Number: ");
                        businessToModify.LicenseNumber = SimpleConvert.ToInt32(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.LicenseIssueDate:
                        // Change the LicenseIssueDate of the business.
                        Console.WriteLine("Current Business License Issue Date: {0}",
                            businessToModify.LicenseIssueDate.ToShortDateString());
                        Console.Write("New Business License Issue Date: ");
                        businessToModify.LicenseIssueDate =
                            SimpleConvert.ToDateTime(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.LicenseExpirDate:
                        // Change the LicenseExpirDate of the business.
                        Console.WriteLine("Current Business License Expiration Date: {0}",
                            businessToModify.LicenseExpirDate.ToShortDateString());
                        Console.Write("New Business License Expiration Date: ");
                        businessToModify.LicenseExpirDate =
                            SimpleConvert.ToDateTime(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.LicenseStatus:
                        // Change the LicenseStatus of the business.
                        Console.WriteLine("Current Business License Status: {0}",
                            businessToModify.LicenseStatus);
                        Console.Write("New Business License Status: ");
                        businessToModify.LicenseStatus = Console.ReadLine();
                        break;

                    case Item.FieldMenuHelper.CouncilDistrict:
                        // Change the CouncilDistrict of the business.
                        Console.WriteLine("Current Council District: {0}",
                            businessToModify.CouncilDistrict);
                        Console.Write("New Council District: ");
                        businessToModify.CouncilDistrict = Console.ReadLine();
                        break;

```

```
                    default:
                        // Catch-all.
                        break;
                }
            }
            else if (itemToModify is Park)
            {
                // Park object. Handle Park object-specific fields.
                Park parkToModify = itemToModify as Park;

                switch (choice)
                {
                    case Item.FieldMenuHelper.FeatureBaseball:
                        // Change the FeatureBaseball of the park.
                        Console.WriteLine("Current # of Baseball Diamonds: {0}",
                            parkToModify.FeatureBaseball);
                        Console.Write("New # of Baseball Diamonds: ");
                        parkToModify.FeatureBaseball = SimpleConvert.ToInt32(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.FeatureBasketball:
                        // Change the FeatureBasketball of the park.
                        Console.WriteLine("Current # of Basketball Courts: {0}",
                            parkToModify.FeatureBasketball);
                        Console.Write("New # of Basketball Courts: ");
                        parkToModify.FeatureBasketball = SimpleConvert.ToSingle(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.FeatureGolf:
                        // Change the Type of the park.
                        Console.WriteLine("Current # of Golf Courses: {0}",
                            parkToModify.FeatureGolf);
                        Console.Write("New # of Golf Courses: ");
                        parkToModify.FeatureGolf = SimpleConvert.ToSingle(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.FeatureLargeMPField:
                        // Change the Type of the park.
                        Console.WriteLine("Current # of Large Multipurpose Fields: {0}",
                            parkToModify.FeatureLargeMPField);
                        Console.Write("New # of Large Multipurpose Fields: ");
                        parkToModify.FeatureLargeMPField =
                            SimpleConvert.ToInt32(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.FeatureTennis:
                        // Change the Type of the park.
                        Console.WriteLine("Current # of Tennis Courts: {0}",
                            parkToModify.FeatureTennis);
                        Console.Write("New # of Tennis Courts: ");
                        parkToModify.FeatureTennis = SimpleConvert.ToInt32(Console.ReadLine());
                        break;

                    case Item.FieldMenuHelper.FeatureVolleyball:
                        // Change the Type of the park.
                        Console.WriteLine("Current # of Volleyball Courts: {0}",
                            parkToModify.FeatureVolleyball);
                        Console.Write("New # of Volleyball Courts: ");
                        parkToModify.FeatureVolleyball = SimpleConvert.ToInt32(Console.ReadLine());
                        break;

                    default:
                        // Catch-all.
                        break;
                }
            }
```

```
            // Modify the item in itemDB.
            itemDB.Modify(itemToModify);

            // Extra line for formatting.
            Console.WriteLine();

            // Return choice so the calling method knows what the choice was and can act
            // accordingly.
            return choice;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    DataSave
         * Type:    Method
         * Purpose: Save the data in itemDB before exiting.
         * Input:   Nothing.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        private void DataSave()
        {
            // Display user's choice.
            Console.WriteLine("-----------------");
            Console.WriteLine("| Save and Exit |");
            Console.WriteLine("-----------------");

            if (itemDB.IsChanged)
            {
                // ItemDB has been changed, ask the user if they wish to save and get response.
                bool validInput;
                Console.Write("Changes detected in the item list, do you wish to save? [Y]/N");
                do
                {
                    validInput = false;
                    ConsoleKeyInfo keyPress = Console.ReadKey(true);
                    switch (keyPress.Key)
                    {
                        case ConsoleKey.Enter:
                            if (keyPress.Modifiers == 0)
                                // User pressed Enter; continue with save.
                                validInput = true;
                            break;

                        case ConsoleKey.Y:
                            if (keyPress.Modifiers == 0 ||
                                keyPress.Modifiers == ConsoleModifiers.Shift)
                                // User pressed 'Y'; continue with save.
                                validInput = true;
                            break;

                        case ConsoleKey.N:
                            if (keyPress.Modifiers == 0 ||
                                keyPress.Modifiers == ConsoleModifiers.Shift)
                            {
                                // User pressed 'N'; abort save.
                                Console.WriteLine();
                                return;
                            }
                            break;

                        default:
                            break;
                    }
                    // Loop while invalid input.
                } while (!validInput);

```

```
                Console.WriteLine("\n\n!!!WARNING!!! Any file you choose will be OVERWRITTEN.");

                string fileNameBusinesses = "";
                string fileNameParks = "";
                string fileNamePublicFacilities = "";
                bool saveSuccess = false;

                // Utilize the search function to create item DBs of each type of item.
                ItemDB allBusinesses = itemDB.Search(
                    "",
                    Enum.GetName(typeof(TypeMenu), TypeMenu.Business).ToLower(),
                    Item.FieldMenuHelper.Name);
                ItemDB allParks = itemDB.Search(
                    "",
                    Enum.GetName(typeof(TypeMenu), TypeMenu.Park).ToLower(),
                    Item.FieldMenuHelper.Name);
                ItemDB allPublicFacilities = itemDB.Search(
                    "",
                    Enum.GetName(typeof(TypeMenu), TypeMenu.PublicFacility).ToLower(),
                    Item.FieldMenuHelper.Name);

                // If the DBs aren't empty, ask for a filename for that item type.
                if (allBusinesses.Count != 0)
                {
                    Console.Write("Please choose a filename for business items: ");
                    fileNameBusinesses = Console.ReadLine();
                }
                if (allParks.Count != 0)
                {
                    Console.Write("Please choose a filename for park items: ");
                    fileNameParks = Console.ReadLine();
                }
                if (allPublicFacilities.Count != 0)
                {
                    Console.Write("Please choose a filename for public facility items: ");
                    fileNamePublicFacilities = Console.ReadLine();
                }

                Console.WriteLine();

                // Attempt to save the business data.
                try
                {
                    if (fileNameBusinesses != "")
                        using (StreamWriter fileWriter = new StreamWriter(fileNameBusinesses))
                        {
                            foreach (var item in allBusinesses)
                                fileWriter.WriteLine(item.ToStringCSV());
                            Console.WriteLine("Business data saved successfully.");
                            saveSuccess = true;
                        }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Error attempting to save business data:");
                    Console.WriteLine(ex.Message);
                }

                // Attempt to save the park data.
                try
                {
                    if (fileNameParks != "")
                        using (StreamWriter fileWriter = new StreamWriter(fileNameParks))
                        {
                            foreach (var item in allParks)
                                fileWriter.WriteLine(item.ToStringCSV());
```

```
                            Console.WriteLine("Park data saved successfully.");
                            saveSuccess = true;
                        }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Error attempting to save park data:");
                    Console.WriteLine(ex.Message);
                }

                // Attempt to save the public facility data.
                try
                {
                    if (fileNamePublicFacilities != "")
                        using (StreamWriter fileWriter = new StreamWriter(fileNamePublicFacilities))
                        {
                            foreach (var item in allPublicFacilities)
                                fileWriter.WriteLine(item.ToStringCSV());
                            Console.WriteLine("Public facility data saved successfully.");
                            saveSuccess = true;
                        }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Error attempting to save public facility data:");
                    Console.WriteLine(ex.Message);
                }

                if (saveSuccess)
                    Console.WriteLine();
            }
            else
            {
                // ItemDB has not been changed.
                Console.WriteLine("No changes to save.\n");
            }
        }

        /*-----------------------------------------------------------------------------------------
         * Name:    DataSearch
         * Type:    Method
         * Purpose: Interactively searches for objects based upon user input.
         * Input:   Nothing.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        private void DataSearch()
        {
            TypeMenu typeChoice;

            do
            {
                // Display the user's choice.
                Console.WriteLine("----------------");
                Console.WriteLine("| Search Items |");
                Console.WriteLine("----------------");
                typeChoice = TypeMenuDisplay();

                if (typeChoice != TypeMenu.Back)
                {
                    do
                    {
                        // Display the user's choice.
                        switch (typeChoice)
                        {
                            case TypeMenu.Business:
                                Console.WriteLine("---------------------");
```

```
                                Console.WriteLine("| Search Businesses |");
                                Console.WriteLine("---------------------");
                                break;

                            case TypeMenu.Park:
                                Console.WriteLine("----------------");
                                Console.WriteLine("| Search Parks |");
                                Console.WriteLine("----------------");
                                break;

                            case TypeMenu.PublicFacility:
                                Console.WriteLine("----------------------------");
                                Console.WriteLine("| Search Public Facilities |");
                                Console.WriteLine("----------------------------");
                                break;

                            case TypeMenu.Back:
                                // Nothing to do; user wants to go back.
                            default:
                                // Catch-all.
                                break;
                        }

                        // Loop while the user has not chosen to go back.
                    } while (DataSearchMenuAction(FieldMenuDisplay(typeChoice), typeChoice) !=
                        Item.FieldMenuHelper.Back);
                }
                // Loop while the user has not chosen to go back.
            } while (typeChoice != TypeMenu.Back);
        }

        /*-----------------------------------------------------------------------------------------
         * Name:    DataSearchMenuAction
         * Type:    Method
         * Purpose: Acts on the user's choice made at the Search Menu.
         * Input:   Item.FieldMenuHelper field, represents the action specified.
         * Input:   TypeMenu type, represents the type of item the user is searching for.
         * Output:  Item.FieldMenuHelper, represents the action specified.
        -----------------------------------------------------------------------------------------*/
        private Item.FieldMenuHelper DataSearchMenuAction(Item.FieldMenuHelper field, TypeMenu type)
        {
            // Decide what to display based on the user's type.
            switch (field)
            {
                case Item.FieldMenuHelper.Name:
                    // Search the Name property.
                    Console.WriteLine("------------------------");
                    Console.WriteLine("| Search Items -- Name |");
                    Console.WriteLine("------------------------");
                    break;

                case Item.FieldMenuHelper.Type:
                    // Search the Type property.
                    Console.WriteLine("------------------------");
                    Console.WriteLine("| Search Items -- Type |");
                    Console.WriteLine("------------------------");
                    break;

                case Item.FieldMenuHelper.StreetAddress:
                    // Search the StreetAddress property.
                    Console.WriteLine("----------------------------------");
                    Console.WriteLine("| Search Items -- Street Address |");
                    Console.WriteLine("----------------------------------");
                    break;

                case Item.FieldMenuHelper.City:
```

```
                    // Search the City property.
                    Console.WriteLine("------------------------");
                    Console.WriteLine("| Search Items -- City |");
                    Console.WriteLine("------------------------");
                    break;

                case Item.FieldMenuHelper.State:
                    // Search the State property.
                    Console.WriteLine("-------------------------");
                    Console.WriteLine("| Search Items -- State |");
                    Console.WriteLine("-------------------------");
                    break;

                case Item.FieldMenuHelper.Zip:
                    // Search the Zip property.
                    Console.WriteLine("----------------------------");
                    Console.WriteLine("| Search Items -- ZIP Code |");
                    Console.WriteLine("----------------------------");
                    break;

                case Item.FieldMenuHelper.Latitude:
                    // Search the Latitude property.
                    Console.WriteLine("----------------------------");
                    Console.WriteLine("| Search Items -- Latitude |");
                    Console.WriteLine("----------------------------");
                    break;

                case Item.FieldMenuHelper.Longitude:
                    // Search the Longitude property.
                    Console.WriteLine("-----------------------------");
                    Console.WriteLine("| Search Items -- Longitude |");
                    Console.WriteLine("-----------------------------");
                    break;

                case Item.FieldMenuHelper.Phone:
                    // Search the Phone property.
                    Console.WriteLine("-------------------------------");
                    Console.WriteLine("| Search Items -- Phone Number |");
                    Console.WriteLine("-------------------------------");
                    break;

                case Item.FieldMenuHelper.LicenseFiscalYear:
                    // Search the LicenseFiscalYear property.
                    Console.WriteLine("-----------------------------------------------");
                    Console.WriteLine("| Search Items -- Business License Fiscal Year |");
                    Console.WriteLine("-----------------------------------------------");
                    break;

                case Item.FieldMenuHelper.LicenseNumber:
                    // Search the LicenseNumber property.
                    Console.WriteLine("-----------------------------------------");
                    Console.WriteLine("| Search Items -- Business License Number |");
                    Console.WriteLine("-----------------------------------------");
                    break;

                case Item.FieldMenuHelper.LicenseIssueDate:
                    // Search the LicenseIssueDate property.
                    Console.WriteLine("-----------------------------------------------");
                    Console.WriteLine("| Search Items -- Business License Issue Date |");
                    Console.WriteLine("-----------------------------------------------");
                    break;

                case Item.FieldMenuHelper.LicenseExpirDate:
                    // Search the LicenseExpirDate property.
                    Console.WriteLine("-------------------------------------------------");
                    Console.WriteLine("| Search Items -- Business License Expiration Date |");
```

```
                    Console.WriteLine("----------------------------------------------------");
                    break;

                case Item.FieldMenuHelper.LicenseStatus:
                    // Search the LicenseStatus property.
                    Console.WriteLine("-----------------------------------------");
                    Console.WriteLine("| Search Items -- Business License Status |");
                    Console.WriteLine("-----------------------------------------");
                    break;

                case Item.FieldMenuHelper.CouncilDistrict:
                    // Search the CouncilDistrict property.
                    Console.WriteLine("-----------------------------------");
                    Console.WriteLine("| Search Items -- Council District |");
                    Console.WriteLine("-----------------------------------");
                    break;

                case Item.FieldMenuHelper.FeatureBaseball:
                    // Search the FeatureBaseball property.
                    Console.WriteLine("----------------------------------------");
                    Console.WriteLine("| Search Items -- # of Baseball Diamonds |");
                    Console.WriteLine("----------------------------------------");
                    break;

                case Item.FieldMenuHelper.FeatureBasketball:
                    // Search the FeatureBasketball property.
                    Console.WriteLine("----------------------------------------");
                    Console.WriteLine("| Search Items -- # of Basketball Courts |");
                    Console.WriteLine("----------------------------------------");
                    break;

                case Item.FieldMenuHelper.FeatureGolf:
                    // Search the FeatureGolf property.
                    Console.WriteLine("-------------------------------------");
                    Console.WriteLine("| Search Items -- # of Golf Courses |");
                    Console.WriteLine("-------------------------------------");
                    break;

                case Item.FieldMenuHelper.FeatureLargeMPField:
                    // Search the FeatureLargeMPField property.
                    Console.WriteLine("--------------------------------------------------");
                    Console.WriteLine("| Search Items -- # of Large Multipurpose Fields |");
                    Console.WriteLine("--------------------------------------------------");
                    break;

                case Item.FieldMenuHelper.FeatureTennis:
                    // Search the FeatureTennis property.
                    Console.WriteLine("-------------------------------------");
                    Console.WriteLine("| Search Items -- # of Tennis Courts |");
                    Console.WriteLine("-------------------------------------");
                    break;

                case Item.FieldMenuHelper.FeatureVolleyball:
                    // Search the FeatureVolleyball property.
                    Console.WriteLine("----------------------------------------");
                    Console.WriteLine("| Search Items -- # of Volleyball Courts |");
                    Console.WriteLine("----------------------------------------");
                    break;

                case Item.FieldMenuHelper.Back:
                case Item.FieldMenuHelper.BackBusiness:
                case Item.FieldMenuHelper.BackPark:
                    // Nothing to do; the user wants to go back.
                default:
                    // Catch-all.
                    return Item.FieldMenuHelper.Back;
```

```
            }

            Console.WriteLine("What kind of comparator do you wish to use?");
            Console.WriteLine("  | - contains (default)    >= - greater than or equal to");
            Console.WriteLine(" !| - does not contain      <= - less than or equal to");
            Console.WriteLine("  = - equal                  > - greater than");
            Console.WriteLine(" != - not equal              < - less than");
            Console.Write("Choice: ");
            string comparator = Console.ReadLine();
            Console.WriteLine();

            // Get the user's search text and pipe that directly into the search method.
            Console.Write("Enter your search text: ");
            ItemDB foundItems = itemDB.Search(Console.ReadLine(),
                Enum.GetName(typeof(TypeMenu), type).ToLower(), field, comparator);

            // Show the results.
            Console.WriteLine("");
            Console.WriteLine("------------------");
            Console.WriteLine("| Search Results |");
            Console.WriteLine("------------------");
            Console.WriteLine("{0} item{1} found.\n", foundItems.Count,
                              foundItems.Count == 1 ? "" : "s");

            // Display any found items.
            foundItems.DisplayAll();

            // Return choice so the calling method knows what the choice was and can act
            // accordingly.
            return field;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    DataStatistics
         * Type:    Method
         * Purpose: Display a count of unique Type fields and display those Type values.
         * Input:   Nothing.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        private void DataStatistics()
        {
            Console.WriteLine("--------------");
            Console.WriteLine("| Statistics |");
            Console.WriteLine("--------------");

            itemDB.Statistics();
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FieldMenuDisplay
         * Type:    Method
         * Purpose: Display the field menu and get a choice. Must have valid input to return.
         * Input:   Item item, used to get the user's choice of item type.
         * Output:  Item.FieldMenuHelper, representing the choice that was made.
        -------------------------------------------------------------------------------------*/
        private Item.FieldMenuHelper FieldMenuDisplay(Item item)
        {
            TypeMenu type;

            // Determine the type of the item.
            if (item is Business)
                type = TypeMenu.Business;
            else if (item is Park)
                type = TypeMenu.Park;
            else if (item is PublicFacility)
                type = TypeMenu.PublicFacility;
```

```
            else
                // Something went wrong; this should never be encountered.
                throw new InvalidOperationException(
                    "Attempted to access field menu for an invalid object.");

            // Call the full field menu and pass through the returned Item.FieldMenuHelper object.
            return FieldMenuDisplay(type);
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FieldMenuDisplay
         * Type:    Method
         * Purpose: Display the field menu and get a choice. Must have valid input to return.
         * Input:   TypeMenu type, contains the user's choice of item type.
         * Output:  Item.FieldMenuHelper, representing the choice that was made.
        -------------------------------------------------------------------------------------*/
        private Item.FieldMenuHelper FieldMenuDisplay(TypeMenu type)
        {
            Item.FieldMenuHelper menuChoice = 0;
            int offset = 0;
            bool invalid = true;

            do
            {
                // Display the common part of the menu.
                Console.WriteLine("Please select the field you would like to work with:");
                Console.WriteLine("  1) Name");
                Console.WriteLine("  2) Type");
                Console.WriteLine("  3) Street Address");
                Console.WriteLine("  4) City");
                Console.WriteLine("  5) State");
                Console.WriteLine("  6) ZIP Code");
                Console.WriteLine("  7) Latitude");
                Console.WriteLine("  8) Longitude");
                Console.WriteLine("  9) Phone Number");

                if (type == TypeMenu.Business)
                {
                    // Display the business-specific part of the menu.
                    Console.WriteLine(" 10) Business License Fiscal Year");
                    Console.WriteLine(" 11) Business License Number");
                    Console.WriteLine(" 12) Business License Issued Date");
                    Console.WriteLine(" 13) Business License Expiration Date");
                    Console.WriteLine(" 14) Business License Status");
                    Console.WriteLine(" 15) Council District");
                    Console.WriteLine(" 16) Back");

                    // Offset used to convert choice to the proper FieldMenuHelper value.
                    offset = Business.FieldOffset;
                }
                else if (type == TypeMenu.Park)
                {
                    // Display the park-specific part of the menu.
                    Console.WriteLine(" 10) Number of Baseball Diamonds");
                    Console.WriteLine(" 11) Number of Basketball Courts");
                    Console.WriteLine(" 12) Number of Golf Courses");
                    Console.WriteLine(" 13) Number of Large Multipurpose FieldMenuHelper");
                    Console.WriteLine(" 14) Number of Tennis Courts");
                    Console.WriteLine(" 15) Number of Volleyball Courts");
                    Console.WriteLine(" 16) Back");

                    // Offset used to convert choice to the proper FieldMenuHelper value.
                    offset = Park.FieldOffset;
                }
                else
                {
```

```
                    // Display the public facility-specific part of the menu.
                    Console.WriteLine(" 10) Back");
                }

                // Ask the user for their choice.
                Console.Write("Choice: ");
                string input = Console.ReadLine();

                // Extra line for formatting.
                Console.WriteLine();

                // Attempts to parse user input, then adjusts menuChoice based on item type and
                // hands it off for actual validation.
                invalid = !Item.FieldMenuHelper.TryParse(input, out menuChoice);
                if (!invalid && menuChoice > Item.FieldCommonMax && type != TypeMenu.PublicFacility)
                    menuChoice += offset;
                invalid = invalid || !FieldMenuValidate(menuChoice, type);
            } while (invalid);

            // Return the user's choice.
            return menuChoice;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FieldMenuValidate
         * Type:    Method
         * Purpose: Validates that the choice by the user is within the limits and is logically
         *          possible.
         * Input:   Item.FieldMenuHelper value, contains the user's choice.
         * Input:   TypeMenu type, contains the user's choice of item type.
         * Output:  bool, representing whether the user's choice was valid or not.
        -------------------------------------------------------------------------------------*/
        private bool FieldMenuValidate(Item.FieldMenuHelper value, TypeMenu type)
        {
            // General check to make sure that the user input is within valid limits.
            if (value < Item.FieldMin || value > Item.FieldMax)
                return false;
            // General check to see if the chosen field is one that is common to all items.
            else if (value >= Item.FieldCommonMin && value <= Item.FieldCommonMax)
                return true;

            // Check whether the chosen field is valid for the given type.
            switch (type)
            {
                case TypeMenu.Business:
                    if (value >= Business.FieldMin && value <= Business.FieldMax)
                        return true;
                    break;
                case TypeMenu.Park:
                    if (value >= Park.FieldMin && value <= Park.FieldMax)
                        return true;
                    break;
                case TypeMenu.PublicFacility:
                    if (value >= PublicFacility.FieldMin && value <= PublicFacility.FieldMax)
                        return true;
                    break;
                default:
                    break;
            }

            // Chosen field is not valid.
            return false;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    MainMenuAction
```

```
         * Type:    Method
         * Purpose: Acts on the user's choice made at the Main Menu.
         * Input:   MainMenu choice, represents the action specified.
         * Output:  MainMenu, represents the action specified.
        ----------------------------------------------------------------------------------------*/
        private MainMenu MainMenuAction(MainMenu choice)
        {
            // Decide what to do based on the user's choice.
            switch (choice)
            {
                case MainMenu.Load:
                    // Clear the ItemDB then load CSV files.
                    DataLoad();
                    break;

                case MainMenu.Add:
                    // Add a new item.
                    DataAdd();
                    break;

                case MainMenu.Modify:
                    // Modify an existing item.
                    DataModify();
                    break;

                case MainMenu.Search:
                    // Search items.
                    DataSearch();
                    break;

                case MainMenu.Delete:
                    // Delete an item.
                    DataDelete();
                    break;

                case MainMenu.DisplayAll:
                    // Display all the items.
                    DataDisplayAll();
                    break;

                case MainMenu.Statistics:
                    // Display
                    DataStatistics();
                    break;

                case MainMenu.Exit:
                    // Save then exit the program.
                    DataSave();
                    Console.WriteLine("Press any key to continue...");
                    Console.ReadKey();
                    break;

                default:
                    // Catch-all.
                    break;
            }

            // Return choice so the calling method knows what the choice was and can act accordingly.
            return choice;
        }

        /*----------------------------------------------------------------------------------------
         * Name:    MainMenuDisplay
         * Type:    Method
         * Purpose: Display the main menu and get a choice. Must have valid input to return.
         * Input:   Nothing.
```

```
         * Output:  MainMenu, representing the choice that was made.
        ----------------------------------------------------------------------------------------*/
        private MainMenu MainMenuDisplay()
        {
            MainMenu menuChoice = 0;
            bool invalid = true;

            do
            {
                // Display the menu.
                Console.WriteLine("-------------------------");
                Console.WriteLine("| Main Interactive Menu |");
                Console.WriteLine("-------------------------");
                Console.WriteLine("Please select an option:");
                Console.WriteLine("  1) Clear List and Load Data");
                Console.WriteLine("  2) Add New Item");
                Console.WriteLine("  3) Modify Item");
                Console.WriteLine("  4) Search Items");
                Console.WriteLine("  5) Delete Item");
                Console.WriteLine("  6) Display All Items");
                Console.WriteLine("  7) Show Statistics");
                Console.WriteLine("  8) Save and Exit");
                Console.Write("Choice: ");

                // Get the user's choice.
                string input = Console.ReadLine();

                // Extra line for formatting.
                Console.WriteLine();

                // Validate the user input.
                invalid = !MainMenu.TryParse(input, out menuChoice) ||
                          !MainMenuValidate(menuChoice);
            } while (invalid);

            // Return the user's choice.
            return menuChoice;
        }

        /*----------------------------------------------------------------------------------------
         * Name:    MainMenuValidate
         * Type:    Method
         * Purpose: Validates that the choice by the user is within the limits and is logically
         *          possible.
         * Input:   MainMenu value, contains the user's choice.
         * Output:  bool, representing whether the user's choice was valid or not.
        ----------------------------------------------------------------------------------------*/
        private bool MainMenuValidate(MainMenu value)
        {
            // Check to make sure that the user input is within valid limits.
            if (value < MainMenuMin || value > MainMenuMax)
                return false;

            // Otherwise, input is good.
            return true;
        }

        /*----------------------------------------------------------------------------------------
         * Name:    TypeMenuDisplay
         * Type:    Method
         * Purpose: Display the type menu and get a choice. Must have valid input to return.
         * Input:   Nothing.
         * Output:  TypeMenu, representing the choice that was made.
        ----------------------------------------------------------------------------------------*/
        private TypeMenu TypeMenuDisplay()
        {
```

```
            TypeMenu menuChoice = 0;
            bool invalid = true;

            do
            {
                // Display the menu.
                Console.WriteLine("Please select an item type:");
                Console.WriteLine("  1) Business");
                Console.WriteLine("  2) Park");
                Console.WriteLine("  3) Public Facility");
                Console.WriteLine("  4) Back");
                Console.Write("Choice: ");

                // Get the user's choice.
                string input = Console.ReadLine();

                // Extra line for formatting.
                Console.WriteLine();

                // Validate the user input.
                invalid = !TypeMenu.TryParse(input, out menuChoice) ||
                          !TypeMenuValidate(menuChoice);
            } while (invalid);

            // Return the user's choice.
            return menuChoice;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    TypeMenuValidate
         * Type:    Method
         * Purpose: Validates that the choice by the user is within the limits and is logically
         *          possible.
         * Input:   TypeMenu value, contains the user's choice.
         * Output:  bool, representing whether the user's choice was valid or not.
        -------------------------------------------------------------------------------------*/
        private bool TypeMenuValidate(TypeMenu value)
        {
            // Check to make sure that the user input is within valid limits.
            if (value < TypeMenuMin || value > TypeMenuMax)
                return false;

            // Otherwise, input is good.
            return true;
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------
 * 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)
```

```
        {
            // Set the item ID to whatever the current key is, increment the key, then add the item.
            item.ItemID = currentItemKey++;
            itemList.Add(item);
            IsChanged = true;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    Delete
         * Type:    Method
         * Purpose: Attempt to delete the Item with the the specified ItemID.
         * Input:   int itemIDToDelete, specifies the ItemID to delete.
         * Output:  bool, represents whether the deletion was successful or not.
        -------------------------------------------------------------------------------------*/
        public bool Delete(int itemIDToDelete)
        {
            try
            {
                itemList.RemoveAt(GetItemIndex(itemIDToDelete));
                IsChanged = true;
                return true;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                return false;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    DisplayAll
         * Type:    Method
         * Purpose: Display a paginated list of all the items in the ItemDB object. Can be a
         *          simplified list or not.
         * Input:   bool simplified, tells the method whether it should display simplified listing
         *          or not.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        public void DisplayAll(bool simplified = false)
        {
            // Helper constants to determine how many lines are going to be used for displaying each
            // type of item.
            const int linesDisplayedPerBusiness = 10;
            const int linesDisplayedPerPark = 12;
            const int linesDisplayedPerPublicFacility = 6;
            const int linesDisplayedSimplified = 4;

            // Variables to help with controlling pagination flow.
            bool displayAll = false;
            int linesToBeDisplayed = 0;
            int linesDisplayed = 0;
            bool validInput = false;
            ConsoleKeyInfo keyPress;

            foreach (var item in itemList)
            {
                // If the user has chosen to display everything, don't both with the other logic.
                if (!displayAll)
                {
                    // Figure out how many lines are about to be displayed.
                    if (simplified)
                        linesToBeDisplayed = linesDisplayedSimplified;
                    else if (item.ItemType == "business")
                        linesToBeDisplayed = linesDisplayedPerBusiness;
                    else if (item.ItemType == "park")
                        linesToBeDisplayed = linesDisplayedPerPark;
```

```
                    else if (item.ItemType == "publicfacility")
                        linesToBeDisplayed = linesDisplayedPerPublicFacility;

                    // If the number of lines about to be displayed will put the displayed number of
                    // lines since last reset at greater than the number of lines available for
                    // display, pause output and ask the user what to do.
                    if (linesDisplayed + linesToBeDisplayed >= Console.WindowHeight - 1)
                    {
                        Console.WriteLine("Enter for next item. Space for next page. " +
                            "Ctrl+Enter for all. Esc to abort.\n");
                        do
                        {
                            // Reset valid input flag and read user input.
                            validInput = false;
                            keyPress = Console.ReadKey(true);

                            switch (keyPress.Key)
                            {
                                case ConsoleKey.Escape:
                                    if (keyPress.Modifiers == 0)
                                        // User pressed Escape key; abort display method.
                                        return;
                                    break;

                                case ConsoleKey.Spacebar:
                                    if (keyPress.Modifiers == 0)
                                    {
                                        // User wishes to display another page; reset the number of
                                        // lines displayed to 0.
                                        linesDisplayed = 0;
                                        validInput = true;
                                    }
                                    break;

                                case ConsoleKey.Enter:
                                    if (keyPress.Modifiers == ConsoleModifiers.Control)
                                    {
                                        // User wishes to display everything.
                                        displayAll = true;
                                        validInput = true;
                                    }
                                    else if (keyPress.Modifiers == 0)
                                    {
                                        // User wishes to display only the next item.
                                        linesDisplayed -= linesToBeDisplayed;
                                        validInput = true;
                                    }
                                    break;

                                default:
                                    break;
                            }
                            // Loop while the user has not provided valid input.
                        } while (!validInput);
                    }
                    // Update the number of lines that have been displayed.
                    linesDisplayed += linesToBeDisplayed;
                }
                // Display the item. Must use the ToString() method, otherwise VS complains that
                // there are no implicit conversions between Item and string types.
                Console.WriteLine("{0}\n", simplified ? item.ToStringSimple() : item.ToString());
            }

            if (itemList.Count == 0)
                Console.WriteLine("No items to display.\n");
        }
```

```

        /*-------------------------------------------------------------------------------------
         * Name:    GetItem
         * Type:    Method
         * Purpose: Get a copy of the item with the specified ItemID.
         * Input:   int itemID, the itemID of the item to get.
         * Output:  Item, contains a copy of the object with itemID, or null if not found.
        -------------------------------------------------------------------------------------*/
        public Item GetItem(int itemID)
        {
            Item tempItem = itemList.Find(i => i.ItemID == itemID);

            if (tempItem is Business)
                return new Business(tempItem as Business);
            else if (tempItem is Park)
                return new Park(tempItem as Park);
            else if (tempItem is PublicFacility)
                return new PublicFacility(tempItem as PublicFacility);
            else
                return null;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    GetItemIndex
         * Type:    Method
         * Purpose: Finds the index of the specified item ID.
         * Input:   int itemID, contains the item ID to search for.
         * Output:  int, contains the index where the item ID can be found.
        -------------------------------------------------------------------------------------*/
        public int GetItemIndex(int itemID)
        {
            return itemList.FindIndex(i => i.ItemID == itemID);
        }

        /*-------------------------------------------------------------------------------------
         * Name:    Modify
         * Type:    Method
         * Purpose: Modifies an Item in the list.
         * Input:   Item item, contains the item that will be matched with and replace the item with
         *          the same ItemID.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        public void Modify(Item item)
        {
            int index = GetItemIndex(item.ItemID);

            // Verify that the ItemID is in the list and that ItemType is the same.
            if (index >= 0 && itemList[index].ItemType == item.ItemType)
                // Verify that the items are of the same type.
                if ((itemList[index] is Business && item is Business) ||
                    (itemList[index] is Park && item is Park) ||
                    (itemList[index] is PublicFacility && item is PublicFacility))
                {
                    // Replace the item reference in the list.
                    itemList[index] = item;
                    IsChanged = true;
                }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    Reset
         * Type:    Method
         * Purpose: Clears the ItemDB, resets currentItemKey, and resets IsChanged.
         * Input:   Nothing.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
```

```
        public void Reset()
        {
            itemList.Clear();
            currentItemKey = KeyStart;
            IsChanged = false;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    Search
         * Type:    Method
         * Purpose: Performs a search based on the comparator on the specified item type and field.
         * Input:   string toSearchFor, contains the string that is being searched for.
         * Input:   string itemType, contains the item type to search through.
         * Input:   Item.FieldMenuHelper field, contains the field to search through.
         * Input:   string comparator, contains the comparator that will be used. Valid choices are
         *          !=, =, <=, >=, <, >, and !|. Everything else does a "contains"-style search.
         * Output:  ItemDB object that contains the results of the search.
        -------------------------------------------------------------------------------------*/
        public ItemDB Search(string toSearchFor, string itemType, Item.FieldMenuHelper field,
            string comparator = "")
        {
            if (itemList.Count == 0)
                return this;

            var ignoreCase = StringComparison.OrdinalIgnoreCase;

            // Create base list and object for ease-of-use inside the switch.
            var typeLimitedList = this.itemList.Where(i => i.ItemType == itemType);
            object baseObject = typeLimitedList.Select(i => i[field]).First();

            switch (comparator)
            {
                case "!=":
                    if (baseObject is DateTime)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (DateTime)i[field] != SimpleConvert.ToDateTime(toSearchFor)).
                            ToList() };
                    else if (baseObject is float)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (float)i[field] != SimpleConvert.ToSingle(toSearchFor)).
                            ToList() };
                    else if (baseObject is int)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (int)i[field] != SimpleConvert.ToInt32(toSearchFor)).
                            ToList() };
                    else
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (string)i[field] != toSearchFor).
                            ToList() };

                case "=":
                    if (baseObject is DateTime)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (DateTime)i[field] == SimpleConvert.ToDateTime(toSearchFor)).
                            ToList() };
                    else if (baseObject is float)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (float)i[field] == SimpleConvert.ToSingle(toSearchFor)).
                            ToList() };
                    else if (baseObject is int)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (int)i[field] == SimpleConvert.ToInt32(toSearchFor)).
                            ToList() };
                    else
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (string)i[field] == toSearchFor).
```

```
                            ToList() };

                case "<=":
                    if (baseObject is DateTime)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (DateTime)i[field] <= SimpleConvert.ToDateTime(toSearchFor)).
                            ToList() };
                    else if (baseObject is float)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (float)i[field] <= SimpleConvert.ToSingle(toSearchFor)).
                            ToList() };
                    else if (baseObject is int)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (int)i[field] <= SimpleConvert.ToInt32(toSearchFor)).
                            ToList() };
                    else
                        Console.WriteLine(
                            "That comparator doesn't work for this field. Switching to |.");
                    break;

                case ">=":
                    if (baseObject is DateTime)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (DateTime)i[field] >= SimpleConvert.ToDateTime(toSearchFor)).
                            ToList() };
                    else if (baseObject is float)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (float)i[field] >= SimpleConvert.ToSingle(toSearchFor)).
                            ToList() };
                    else if (baseObject is int)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (int)i[field] >= SimpleConvert.ToInt32(toSearchFor)).
                            ToList() };
                    else
                        Console.WriteLine(
                            "That comparator doesn't work for this field. Switching to |.");
                    break;

                case "<":
                    if (baseObject is DateTime)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (DateTime)i[field] < SimpleConvert.ToDateTime(toSearchFor)).
                            ToList() };
                    else if (baseObject is float)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (float)i[field] < SimpleConvert.ToSingle(toSearchFor)).
                            ToList() };
                    else if (baseObject is int)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (int)i[field] < SimpleConvert.ToInt32(toSearchFor)).
                            ToList() };
                    else
                        Console.WriteLine(
                            "That comparator doesn't work for this field. Switching to |.");
                    break;

                case ">":
                    if (baseObject is DateTime)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (DateTime)i[field] > SimpleConvert.ToDateTime(toSearchFor)).
                            ToList() };
                    else if (baseObject is float)
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (float)i[field] > SimpleConvert.ToSingle(toSearchFor)).
                            ToList() };
                    else if (baseObject is int)
```

```
                        return new ItemDB() { itemList = typeLimitedList.
                            Where(i => (int)i[field] > SimpleConvert.ToInt32(toSearchFor)).
                            ToList() };
                    else
                        Console.WriteLine(
                            "That comparator doesn't work for this field. Switching to |.");
                    break;

                case "!|":
                    return new ItemDB() { itemList = typeLimitedList.
                        Where(i => i[field].ToString().IndexOf(toSearchFor, ignoreCase) < 0).
                        ToList() };

                default:
                    break;
            }

            // Default/catch-all search.
            return new ItemDB() { itemList = typeLimitedList.
                Where(i => i[field].ToString().IndexOf(toSearchFor, ignoreCase) >= 0).
                ToList() };
        }

        /*-----------------------------------------------------------------------------------------
         * Name:    Statistics
         * Type:    Method
         * Purpose: Displays the number of unique Type fields, and then displays the field values
         *          and their count.
         * Input:   Nothing.
         * Output:  Nothing.
        -----------------------------------------------------------------------------------------*/
        public void Statistics()
        {
            // Create a Dictionary to keep track of the unique Item.Type values.
            Dictionary<string, int> types = new Dictionary<string, int>();

            // Get a sorted lowercase list of unique Item.Type values and Add the aforementioned
            // list to the dictionary.
            var uniqueTypes = itemList.Select(i => i.Type.ToLower()).Distinct().OrderBy(s => s);
            foreach (var type in uniqueTypes)
                types.Add(type, 0);

            // Run through the list and increment the count of any type when it is encountered, then
            // display all the results.
            foreach (var item in itemList)
                types[item.Type.ToLower()]++;
            Console.WriteLine("{0} unique type{1} of item{1} found.\n", types.Count,
                types.Count != 1 ? "s" : "");
            foreach (var type in types)
                Console.WriteLine("{0}: {1}", type.Key, type.Value);
            if (types.Count > 0)
                Console.WriteLine();
        }

        // Implementation for the GetEnumerator method. Source:
        // https://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.110).aspx
        IEnumerator IEnumerable.GetEnumerator()
        {
            return (IEnumerator)GetEnumerator();
        }

        public ItemDBEnum GetEnumerator()
        {
            return new ItemDBEnum(itemList);
        }
    }
```

```
    public class ItemDBEnum : IEnumerator
    {
        // Enumerator for the ItemDB class. Much help came from MSDN.
        // https://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.110).aspx

        private List<Item> itemList;

        int position = -1;

        public ItemDBEnum(List<Item> list)
        {
            itemList = list;
        }

        object IEnumerator.Current
        {
            get
            {
                return Current;
            }
        }

        public Item Current
        {
            get
            {
                try
                {
                    return itemList[position];
                }
                catch (IndexOutOfRangeException)
                {
                    throw new InvalidOperationException();
                }
            }
        }

        public bool MoveNext()
        {
            position++;
            return (position < itemList.Count);
        }

        public void Reset()
        {
            position = -1;
        }
    }
}
```

```
/*----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy and Dr. Raman Adaikkalavan
 * Date:        2015-06-17
 * Assignment:  cView-P3
 * Source File: Item.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Provides the base abstract class for data items along with some supporting methods.
----------------------------------------------------------------------------------------------*/

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

namespace Ph3
{
    public abstract class Item
    {
        /*----------------------------------------------------------------------------------------
         * Type:    Helper Constants
        ----------------------------------------------------------------------------------------*/
        public const FieldMenuHelper FieldCommonMin = FieldMenuHelper.Name;
        public const FieldMenuHelper FieldCommonMax = FieldMenuHelper.Phone;
        public const FieldMenuHelper FieldMin = FieldMenuHelper.Name;
        public const FieldMenuHelper FieldMax = FieldMenuHelper.BackPark;

        /*----------------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Basic no-parameter constructor.
         * Input:   Nothing.
        ----------------------------------------------------------------------------------------*/
        public Item()
        {
            // Nothing else to do.
        }

        /*----------------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Copy constructor.
         * Input:   Item fromItem, reference to the other Item from which fields should be copied.
        ----------------------------------------------------------------------------------------*/
        public Item(Item fromItem)
        {
            ItemID = fromItem.ItemID;

            Name = fromItem.Name;
            Type = fromItem.Type;
            StreetAddress = fromItem.StreetAddress;
            City = fromItem.City;
            State = fromItem.State;
            Zip = fromItem.Zip;
            Latitude = fromItem.Latitude;
            Longitude = fromItem.Longitude;
            Phone = fromItem.Phone;
        }

        /*----------------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Constructor that will fill all the properties except ItemID and ItemType.
         * Input:   string name, contains the desired Name for the object.
         * Input:   string type, contains the desired Type for the object.
         * Input:   string streetAddress, contains the desired StreetAddress for the object.
         * Input:   string city, contains the desired City for the object.
```

```
         * Input:   string state, contains the desired State for the object.
         * Input:   string zip, contains the desired Zip for the object.
         * Input:   string latitude, contains the desired Latitude for the object.
         * Input:   string longitude, contains the desired Longitude for the object.
         * Input:   string phone, contains the desired Phone for the object.
        ----------------------------------------------------------------------------------------*/
        public Item(string name, string type, string streetAddress, string city, string state,
            string zip, string latitude, string longitude, string phone)
        {
            Name = name;
            Type = type;
            StreetAddress = streetAddress;
            City = city;
            State = state;
            Zip = zip;
            Latitude = latitude;
            Longitude = longitude;
            Phone = phone;
        }

        /*----------------------------------------------------------------------------------------
         * Name:    FieldMenuHelper
         * Type:    Enum
         * Purpose: Represents the fields in use in this class, with additions for its derived
         *          classes.
        ----------------------------------------------------------------------------------------*/
        public enum FieldMenuHelper
        {
            // Common Fields
            Name = 1,
            Type,
            StreetAddress,
            City,
            State,
            Zip,
            Latitude,
            Longitude,
            Phone,
            Back,

            // Business Fields
            LicenseFiscalYear,
            LicenseNumber,
            LicenseIssueDate,
            LicenseExpirDate,
            LicenseStatus,
            CouncilDistrict,
            BackBusiness,

            // Park Fields
            FeatureBaseball,
            FeatureBasketball,
            FeatureGolf,
            FeatureLargeMPField,
            FeatureTennis,
            FeatureVolleyball,
            BackPark
        }

        /*----------------------------------------------------------------------------------------
         * BEGIN UNTOUCHABLE CODE -->
        ----------------------------------------------------------------------------------------*/
        // Create an ID for each item. So if you have 10 parks and 5 businesses, ID will be 1 to 15.
        public abstract int ItemID { get; set; }

        // Value will be "business", "park", or "publicfacility".
```

```
        public abstract string ItemType { get; set; }

        // Populate from CSV
        public abstract string Name { get; set; }
        public abstract string Type { get; set; }
        public abstract string StreetAddress { get; set; }
        public abstract string City { get; set; }
        public abstract string State { get; set; }
        public abstract string Zip { get; set; }
        public abstract string Latitude { get; set; }
        public abstract string Longitude { get; set; }
        public abstract string Phone { get; set; }
        /*-----------------------------------------------------------------------------------------
         * <-- END UNTOUCHABLE CODE
        -----------------------------------------------------------------------------------------*/

        /*-----------------------------------------------------------------------------------------
         * Name:    this[]
         * Type:    Indexer
         * Purpose: Provides easy access to the properties of the class. Need to change the indexer
         *          name because its default is "Item" and the compiler throws a fit because the
         *          class is already named that.
         * Input:   FieldMenuHelper fiendNum, represents the desired property.
         * Output:  object, contains whichever property was desired, or 0 if the property was not
         *          found.
        -----------------------------------------------------------------------------------------*/
        [IndexerName("Index")]
        public virtual object this[FieldMenuHelper fieldNum]
        {
            get
            {
                switch (fieldNum)
                {
                    case FieldMenuHelper.Name:
                        return Name;
                    case FieldMenuHelper.Type:
                        return Type;
                    case FieldMenuHelper.StreetAddress:
                        return StreetAddress;
                    case FieldMenuHelper.City:
                        return City;
                    case FieldMenuHelper.State:
                        return State;
                    case FieldMenuHelper.Zip:
                        return Zip;
                    case FieldMenuHelper.Latitude:
                        return Latitude;
                    case FieldMenuHelper.Longitude:
                        return Longitude;
                    case FieldMenuHelper.Phone:
                        return Phone;
                    default:
                        return 0;
                }
            }
        }

        /*-----------------------------------------------------------------------------------------
         * Name:    ToStringCSV
         * Type:    Method
         * Purpose: Serializes the data contained in the object into a comma-separated value string.
         * Input:   Nothing.
         * Output:  string, representing the data of this object as serialized to a CSV string.
        -----------------------------------------------------------------------------------------*/
        public virtual string ToStringCSV()
        {
```

```
            char separator = ',';
            return Name + separator + Type + separator + StreetAddress + separator + City +
                separator + State +separator + Zip + separator + Latitude + separator + Longitude +
                separator + Phone;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ToStringSimple
         * Type:    Method
         * Purpose: Formats the data contained in the object into a simplified string containing
         *          only the ItemID, ItemType, and Name properties.
         * Input:   Nothing.
         * Output:  string, representing a simplified view of this object.
        -------------------------------------------------------------------------------------*/
        public virtual string ToStringSimple()
        {
            // Returns a string formatted as follows:
            //   Item ID: <ItemID>
            // Item Type: <ItemType>
            //      Name: <Name>
            return string.Format(
                "  Item ID: {0}\n" +
                "Item Type: {1}\n" +
                "     Name: {2}",
                ItemID,
                ItemType,
                Name);
        }
    }
}
```

```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-17
 * Assignment:  cView-P3
 * Source File: Business.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Contains the Business class, derived from the Item abstract class, and supporting
 *              methods.
-----------------------------------------------------------------------------------------*/

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

namespace Ph3
{
    public class Business : Item
    {
        /*-----------------------------------------------------------------------------------
         * Type:    Helper Constants
        -----------------------------------------------------------------------------------*/
        public new const FieldMenuHelper FieldMin = FieldMenuHelper.LicenseFiscalYear;
        public new const FieldMenuHelper FieldMax = FieldMenuHelper.BackBusiness;
        public const int FieldOffset = 1;

        /*-----------------------------------------------------------------------------------
         * Type:    Private Fields
        -----------------------------------------------------------------------------------*/
        private string itemType = "business";
        private int licenseFiscalYear;
        private int licenseNumber;

        /*-----------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Basic no-parameter constructor.
         * Input:   Nothing.
        -----------------------------------------------------------------------------------*/
        public Business()
        {
            // Nothing else to do.
        }

        /*-----------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Copy constructor.
         * Input:   Business fromItem, reference to the other Business from which fields should be
         *          copied.
        -----------------------------------------------------------------------------------*/
        public Business(Business fromItem)
            : base(fromItem)
        {
            itemType = fromItem.itemType;

            LicenseFiscalYear = fromItem.LicenseFiscalYear;
            LicenseNumber = fromItem.LicenseNumber;
            LicenseIssueDate = fromItem.LicenseIssueDate;
            LicenseExpirDate = fromItem.LicenseExpirDate;
            LicenseStatus = fromItem.LicenseStatus;
            CouncilDistrict = fromItem.CouncilDistrict;
        }

        /*-----------------------------------------------------------------------------------
         * Type:    Constructor
```

```
         * Purpose: Constructor that will fill all the properties except ItemID and ItemType.
         * Input:   string name, contains the desired Name for the object.
         * Input:   string type, contains the desired Type for the object.
         * Input:   string streetAddress, contains the desired StreetAddress for the object.
         * Input:   string city, contains the desired City for the object.
         * Input:   string state, contains the desired State for the object.
         * Input:   string zip, contains the desired Zip for the object.
         * Input:   string latitude, contains the desired Latitude for the object.
         * Input:   string longitude, contains the desired Longitude for the object.
         * Input:   string phone, contains the desired Phone for the object.
         * Input:   int licenseFiscalYear, contains the desired LicenseFiscalYear for the object.
         * Input:   int licenseNumber, contains the desired LicenseNumber for the object.
         * Input:   DateTime licenseIssueDate, contains the desired LicenseIssueDate for the object.
         * Input:   DateTime licenseExpirDate, contains the desired LicenseExpirDate for the object.
         * Input:   string licenseStatus, contains the desired LicenseStatus for the object.
         * Input:   string councilDistrict, contains the desired CouncilDistrict for the object.
         * Output:  Nothing.
        ----------------------------------------------------------------------------------------*/
        public Business(string name, string type, string streetAddress, string city, string state,
            string zip, string latitude, string longitude, string phone, int licenseFiscalYear,
            int licenseNumber, DateTime licenseIssueDate, DateTime licenseExpirDate,
            string licenseStatus, string councilDistrict)
            : base(name, type, streetAddress, city, state, zip, latitude, longitude, phone)
        {
            LicenseFiscalYear = licenseFiscalYear;
            LicenseNumber = licenseNumber;
            LicenseIssueDate = licenseIssueDate;
            LicenseExpirDate = licenseExpirDate;
            LicenseStatus = licenseStatus;
            CouncilDistrict = councilDistrict;
        }

        /*----------------------------------------------------------------------------------------
         * Type:    Auto-implemented Properties
        ----------------------------------------------------------------------------------------*/
        public override int ItemID { get; set; }            // Item ID

        public override string Name { get; set; }           // Business Name
        public override string Type { get; set; }           // Business Classification
        public override string StreetAddress { get; set; }  // Street Address
        public override string City { get; set; }           // City
        public override string State { get; set; }          // State
        public override string Zip { get; set; }            // Zip Code
        public override string Latitude { get; set; }       // Latitude
        public override string Longitude { get; set; }      // Longitude
        public override string Phone { get; set; }          // Phone Number

        public DateTime LicenseIssueDate { get; set; }      // Business License Issue Date
        public DateTime LicenseExpirDate { get; set; }      // Business License Expiration Date
        public string LicenseStatus { get; set; }           // Business License Status
        public string CouncilDistrict { get; set; }         // Council District

        /*----------------------------------------------------------------------------------------
         * Name:    ItemType
         * Type:    Property
         * Purpose: Provides access to the itemType field.
        ----------------------------------------------------------------------------------------*/
        public override string ItemType
        {
            get
            {
                return itemType;
            }
            set
            {
                // Do nothing.
```

```
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    LicenseFiscalYear
         * Type:    Property
         * Purpose: Provides access to the licenseFiscalYear field, and validation for the same.
        -------------------------------------------------------------------------------------*/
        public int LicenseFiscalYear
        {
            get
            {
                return licenseFiscalYear;
            }
            set
            {
                if (value >= 0)
                    licenseFiscalYear = value;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    LicenseNumber
         * Type:    Property
         * Purpose: Provides access to the licenseNumber field, and validation for the same.
        -------------------------------------------------------------------------------------*/
        public int LicenseNumber
        {
            get
            {
                return licenseNumber;
            }
            set
            {
                if (value >= 0)
                    licenseNumber = value;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    this[]
         * Type:    Indexer
         * Purpose: Provides easy access to the properties of the class.
         * Input:   FieldMenuHelper fiendNum, represents the desired property.
         * Output:  object, contains whichever property was desired, or 0 if the property was not
         *          found.
        -------------------------------------------------------------------------------------*/
        public override object this[FieldMenuHelper fieldNum]
        {
            get
            {
                switch (fieldNum)
                {
                    case FieldMenuHelper.LicenseFiscalYear:
                        return LicenseFiscalYear;
                    case FieldMenuHelper.LicenseNumber:
                        return LicenseNumber;
                    case FieldMenuHelper.LicenseIssueDate:
                        return LicenseIssueDate;
                    case FieldMenuHelper.LicenseExpirDate:
                        return LicenseExpirDate;
                    case FieldMenuHelper.LicenseStatus:
                        return LicenseStatus;
                    case FieldMenuHelper.CouncilDistrict:
                        return CouncilDistrict;
                    default:
```

```
                        return base[fieldNum];
                    }
                }
            }

            /*-----------------------------------------------------------------------------------------
             * Name:    ToStringCSV
             * Type:    Method
             * Purpose: Serializes the data contained in the object into a comma-separated value string.
             * Input:   Nothing.
             * Output:  string, representing the data of this object as serialized to a CSV string.
            -----------------------------------------------------------------------------------------*/
            public override string ToStringCSV()
            {
                char separator = ',';
                return base.ToStringCSV() + separator + LicenseFiscalYear + '-' + LicenseNumber +
                    separator + LicenseIssueDate.ToShortDateString() + separator +
                    LicenseExpirDate.ToShortDateString() + separator + LicenseStatus + separator +
                    CouncilDistrict;
            }

            /*-----------------------------------------------------------------------------------------
             * Name:    ToString
             * Type:    Method
             * Purpose: Override of ToString() method. Formats the data contained in this object so it
             *          looks pretty.
             * Input:   Nothing.
             * Output:  string, containing serialized object data.
            -----------------------------------------------------------------------------------------*/
            public override string ToString()
            {
                // Returns a string formatted as follows:
                //  Item ID (Item Type): <ItemID> (<ItemType>)
                // Business Name (Type): <Name> (<Type>)
                //              Address: <StreetAddress>, <City>, <State> <Zip>
                //      GPS Coordinates: (<Latitude>, <Longitude>)
                //         Phone Number: <Phone>
                //       License Number: <LicenseFiscalYear>-<LicenseNumber>
                //                Valid: From <LicenseIssueDate> to <LicenseExpirDate>
                //               Status: <LicenseStatus>
                //     Council District: <CouncilDistrict>
                return string.Format(
                    " Item ID (Item Type): {0} ({1})\n" +
                    "Business Name (Type): {2} ({3})\n" +
                    "              Address: {4}, {5}, {6} {7}\n" +
                    "      GPS Coordinates: ({8}, {9})\n" +
                    "         Phone Number: {10}\n" +
                    "       License Number: {11}-{12}\n" +
                    "                Valid: From {13} to {14}\n" +
                    "               Status: {15}\n" +
                    "     Council District: {16}",
                    ItemID, ItemType,
                    Name, Type,
                    StreetAddress, City, State, Zip,
                    Latitude, Longitude,
                    Phone,
                    LicenseFiscalYear, LicenseNumber,
                    LicenseIssueDate.ToShortDateString(), LicenseExpirDate.ToShortDateString(),
                    LicenseStatus,
                    CouncilDistrict);
            }
        }
}
```

```
/*----------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy and Dr. Raman Adaikkalavan
 * Date:        2015-06-17
 * Assignment:  cView-P3
 * Source File: Park.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Contains the Park class, derived from the Item abstract class, and supporting
 *              methods.
----------------------------------------------------------------------------------------------*/

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

namespace Ph3
{
    public class Park : Item
    {
        /*----------------------------------------------------------------------------------------
         * Type:    Helper Constants
        ----------------------------------------------------------------------------------------*/
        public new const FieldMenuHelper FieldMin = FieldMenuHelper.FeatureBaseball;
        public new const FieldMenuHelper FieldMax = FieldMenuHelper.BackPark;
        public const int FieldOffset = 8;

        /*----------------------------------------------------------------------------------------
         * Type:    Private Fields
        ----------------------------------------------------------------------------------------*/
        private string itemType = "park";
        private int featureBaseball;
        private float featureBasketball;
        private float featureGolf;
        private int featureLargeMPField;
        private int featureTennis;
        private int featureVolleyball;

        /*----------------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Basic no-parameter constructor.
         * Input:   Nothing.
        ----------------------------------------------------------------------------------------*/
        public Park()
        {
            // Nothing else to do.
        }

        /*----------------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Copy constructor.
         * Input:   Park fromItem, reference to the other Park from which fields should be copied.
        ----------------------------------------------------------------------------------------*/
        public Park(Park fromItem)
            : base(fromItem)
        {
            itemType = fromItem.itemType;

            FeatureBaseball = fromItem.FeatureBaseball;
            FeatureBasketball = fromItem.FeatureBasketball;
            FeatureGolf = fromItem.FeatureGolf;
            FeatureLargeMPField = fromItem.FeatureLargeMPField;
            FeatureTennis = fromItem.FeatureTennis;
            FeatureVolleyball = fromItem.FeatureVolleyball;
        }
```

```

        /*-------------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Constructor that will fill all the properties except ItemID and ItemType.
         * Input:   string name, contains the desired Name for the object.
         * Input:   string type, contains the desired Type for the object.
         * Input:   string streetAddress, contains the desired StreetAddress for the object.
         * Input:   string city, contains the desired City for the object.
         * Input:   string state, contains the desired State for the object.
         * Input:   string zip, contains the desired Zip for the object.
         * Input:   string latitude, contains the desired Latitude for the object.
         * Input:   string longitude, contains the desired Longitude for the object.
         * Input:   string phone, contains the desired Phone for the object.
         * Input:   int featureBaseball, contains the desired FeatureBaseball for the object.
         * Input:   float featureBasketball, contains the desired FeatureBasketball for the object.
         * Input:   float featureGolf, contains the desired FeatureGolf for the object.
         * Input:   int featureLargeMPField, contains the desired FeatureLargeMPField for the
         *          object.
         * Input:   int featureTennis, contains the desired FeatureTennis for the object.
         * Input:   int featureVolleyball, contains the desired FeatureVolleyball for the object.
         * Output:  Nothing.
        -------------------------------------------------------------------------------------*/
        public Park(string name, string type, string streetAddress, string city, string state,
            string zip, string latitude, string longitude, string phone, int featureBaseball,
            float featureBasketball, float featureGolf, int featureLargeMPField,
            int featureTennis, int featureVolleyball)
            : base(name, type, streetAddress, city, state, zip, latitude, longitude, phone)
        {
            FeatureBaseball = featureBaseball;
            FeatureBasketball = featureBasketball;
            FeatureGolf = featureGolf;
            FeatureLargeMPField = featureLargeMPField;
            FeatureTennis = featureTennis;
            FeatureVolleyball = featureVolleyball;
        }

        /*-------------------------------------------------------------------------------------
         * Type:    Auto-implemented Properties
        -------------------------------------------------------------------------------------*/
        public override int ItemID { get; set; }            // Item ID

        public override string Name { get; set; }           // Park Name
        public override string Type { get; set; }           // Park Type
        public override string StreetAddress { get; set; }  // Street Address
        public override string City { get; set; }           // City
        public override string State { get; set; }          // State
        public override string Zip { get; set; }            // Zip Code
        public override string Latitude { get; set; }       // Latitude
        public override string Longitude { get; set; }      // Longitude
        public override string Phone { get; set; }          // Phone Number

        /*-------------------------------------------------------------------------------------
         * Name:    ItemType
         * Type:    Property
         * Purpose: Provides access to the itemType field.
        -------------------------------------------------------------------------------------*/
        public override string ItemType
        {
            get
            {
                return itemType;
            }
            set
            {
                // Do nothing.
            }
```

```
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FeatureBaseball
         * Type:    Property
         * Purpose: Provides access to the featureBaseball field, and validation for the same.
        -------------------------------------------------------------------------------------*/
        public int FeatureBaseball
        {
            get
            {
                return featureBaseball;
            }
            set
            {
                if (value >= 0)
                    featureBaseball = value;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FeatureBasketball
         * Type:    Property
         * Purpose: Provides access to the featureBasketball field, and validation for the same.
        -------------------------------------------------------------------------------------*/
        public float FeatureBasketball
        {
            get
            {
                return featureBasketball;
            }
            set
            {
                if (value >= 0 && value % 0.5 == 0)
                    featureBasketball = value;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FeatureGolf
         * Type:    Property
         * Purpose: Provides access to the featureGolf field, and validation for the same.
        -------------------------------------------------------------------------------------*/
        public float FeatureGolf
        {
            get
            {
                return featureGolf;
            }
            set
            {
                if (value >= 0 && value % 0.5 == 0)
                    featureGolf = value;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FeatureLargeMPField
         * Type:    Property
         * Purpose: Provides access to the featureLargeMPField field, and validation for the same.
        -------------------------------------------------------------------------------------*/
        public int FeatureLargeMPField
        {
            get
            {
                return featureLargeMPField;
```

```
            }
            set
            {
                if (value >= 0)
                    featureLargeMPField = value;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FeatureTennis
         * Type:    Property
         * Purpose: Provides access to the featureTennis field, and validation for the same.
        -------------------------------------------------------------------------------------*/
        public int FeatureTennis
        {
            get
            {
                return featureTennis;
            }
            set
            {
                if (value >= 0)
                    featureTennis = value;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    FeatureVolleyball
         * Type:    Property
         * Purpose: Provides access to the featureVolleyball field, and validation for the same.
        -------------------------------------------------------------------------------------*/
        public int FeatureVolleyball
        {
            get
            {
                return featureVolleyball;
            }
            set
            {
                if (value >= 0)
                    featureVolleyball = value;
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    this[]
         * Type:    Indexer
         * Purpose: Provides easy access to the properties of the class.
         * Input:   FieldMenuHelper fiendNum, represents the desired property.
         * Output:  object, contains whichever property was desired, or 0 if the property was not
         *          found.
        -------------------------------------------------------------------------------------*/
        public override object this[FieldMenuHelper fieldNum]
        {
            get
            {
                switch (fieldNum)
                {
                    case FieldMenuHelper.FeatureBaseball:
                        return FeatureBaseball;
                    case FieldMenuHelper.FeatureBasketball:
                        return FeatureBasketball;
                    case FieldMenuHelper.FeatureGolf:
                        return FeatureGolf;
                    case FieldMenuHelper.FeatureLargeMPField:
                        return FeatureLargeMPField;
```

```
                    case FieldMenuHelper.FeatureTennis:
                        return FeatureTennis;
                    case FieldMenuHelper.FeatureVolleyball:
                        return FeatureVolleyball;
                    default:
                        return base[fieldNum];
                }
            }
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ToStringCSV
         * Type:    Method
         * Purpose: Serializes the data contained in the object into a comma-separated value string.
         * Input:   Nothing.
         * Output:  string, representing the data of this object as serialized to a CSV string.
        -------------------------------------------------------------------------------------*/
        public override string ToStringCSV()
        {
            char separator = ',';
            return base.ToStringCSV() + separator + FeatureBaseball + separator +
                FeatureBasketball + separator + FeatureGolf + separator + FeatureLargeMPField +
                separator + FeatureTennis + separator + FeatureVolleyball;
        }

        /*-------------------------------------------------------------------------------------
         * Name:    ToString
         * Type:    Method
         * Purpose: Override of ToString() method. Formats the data contained in this object so it
         *          looks pretty.
         * Input:   Nothing.
         * Output:  string, containing serialized object data.
        -------------------------------------------------------------------------------------*/
        public override string ToString()
        {
            // Returns a the base a string formatted as follows:
            //  Item ID (Item Type): <ItemID> (<ItemType>)
            //     Park Name (Type): <Name> (<Type>)
            //              Address: <StreetAddress>, <City>, <State> <Zip>
            //      GPS Coordinates: (<Latitude>, <Longitude>)
            //         Phone Number: <Phone>
            //    Baseball Diamonds: <FeatureBaseball>
            //    Basketball Courts: <FeatureBasketball>
            //         Golf Courses: <FeatureGolf>
            //      Large MP Fields: <FeatureLargeMPField>
            //        Tennis Courts: <FeatureTennis>
            //    Volleyball Courts: <FeatureVolleyball>
            return string.Format(
                " Item ID (Item Type): {0} ({1})\n" +
                "    Park Name (Type): {2} ({3})\n" +
                "             Address: {4}, {5}, {6} {7}\n" +
                "     GPS Coordinates: ({8}, {9})\n" +
                "        Phone Number: {10}\n" +
                "   Baseball Diamonds: {11}\n" +
                "   Basketball Courts: {12}\n" +
                "        Golf Courses: {13}\n" +
                "     Large MP Fields: {14}\n" +
                "       Tennis Courts: {15}\n" +
                "   Volleyball Courts: {16}",
                ItemID, ItemType,
                Name, Type,
                StreetAddress, City, State, Zip,
                Latitude, Longitude,
                Phone,
                FeatureBaseball,
                FeatureBasketball,
```

```
                FeatureGolf,
                FeatureLargeMPField,
                FeatureTennis,
                FeatureVolleyball);
        }
    }
}

```

```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-17
 * Assignment:  cView-P3
 * Source File: PublicFacility.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Contains the PublicFacility class, derived from the Item abstract class, and
 *              supporting methods.
-----------------------------------------------------------------------------------------*/

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

namespace Ph3
{
    public class PublicFacility : Item
    {
        /*-----------------------------------------------------------------------------------
         * Type:    Helper Constants
         * Purpose:
        -----------------------------------------------------------------------------------*/
        public new const FieldMenuHelper FieldMin = FieldMenuHelper.Name;
        public new const FieldMenuHelper FieldMax = FieldMenuHelper.Back;
        public const int FieldOffset = 0;

        /*-----------------------------------------------------------------------------------
         * Type:    Private Fields
        -----------------------------------------------------------------------------------*/
        private string itemType = "publicfacility";

        /*-----------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Basic no-parameter constructor.
         * Input:   Nothing.
        -----------------------------------------------------------------------------------*/
        public PublicFacility()
        {
            // Nothing else to do.
        }

        /*-----------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Copy constructor.
         * Input:   PublicFacility fromItem, reference to the other PublicFacility from which fields
         *          should be copied.
        -----------------------------------------------------------------------------------*/
        public PublicFacility(PublicFacility fromItem)
            : base(fromItem)
        {
            itemType = fromItem.itemType;
        }

        /*-----------------------------------------------------------------------------------
         * Type:    Constructor
         * Purpose: Constructor that will fill all the properties except ItemID and ItemType.
         * Input:   string name, contains the desired Name for the object.
         * Input:   string type, contains the desired Type for the object.
         * Input:   string streetAddress, contains the desired StreetAddress for the object.
         * Input:   string city, contains the desired City for the object.
         * Input:   string state, contains the desired State for the object.
         * Input:   string zip, contains the desired Zip for the object.
         * Input:   string latitude, contains the desired Latitude for the object.
```

```
         * Input:   string longitude, contains the desired Longitude for the object.
         * Input:   string phone, contains the desired Phone for the object.
         * Output:  Nothing.
        ----------------------------------------------------------------------------------------*/
        public PublicFacility(string name, string type, string streetAddress, string city, string state
,
            string zip, string latitude, string longitude, string phone)
            : base(name, type, streetAddress, city, state, zip, latitude, longitude, phone)
        {
            // Nothing else to do.
        }

        /*----------------------------------------------------------------------------------------
         * Type:    Auto-implemented Properties
        ----------------------------------------------------------------------------------------*/
        public override int ItemID { get; set; }            // Item ID

        public override string Name { get; set; }           // Facility Name
        public override string Type { get; set; }           // Facility Type
        public override string StreetAddress { get; set; }  // Street Address
        public override string City { get; set; }           // City
        public override string State { get; set; }          // State
        public override string Zip { get; set; }            // Zip Code
        public override string Latitude { get; set; }       // Latitude
        public override string Longitude { get; set; }      // Longitude
        public override string Phone { get; set; }          // Phone Number

        /*----------------------------------------------------------------------------------------
         * Name:    ItemType
         * Type:    Property
         * Purpose: Provides access to the itemType field.
        ----------------------------------------------------------------------------------------*/
        public override string ItemType
        {
            get
            {
                return itemType;
            }
            set
            {
                // Do nothing.
            }
        }

        /*----------------------------------------------------------------------------------------
         * Name:    ToString
         * Type:    Method
         * Purpose: Override of ToString() method. Formats the data contained in this object so it
         *          looks pretty.
         * Input:   Nothing.
         * Output:  string, containing serialized object data.
        ----------------------------------------------------------------------------------------*/
        public override string ToString()
        {
            // Returns a the base a string formatted as follows:
            //  Item ID (Item Type): <ItemID> (<ItemType>)
            // Facility Name (Type): <Name> (<Type>)
            //              Address: <StreetAddress>, <City>, <State> <Zip>
            //      GPS Coordinates: (<Latitude>, <Longitude>)
            //         Phone Number: <Phone>
            return string.Format(
                " Item ID (Item Type): {0} ({1})\n" +
                "Facility Name (Type): {2} ({3})\n" +
                "              Address: {4}, {5}, {6} {7}\n" +
                "     GPS Coordinates: ({8}, {9})\n" +
                "        Phone Number: {10}",
```

```
                ItemID, ItemType,
                Name, Type,
                StreetAddress, City, State, Zip,
                Latitude, Longitude,
                Phone);
        }

    }
}

```

```
/*-----------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-06-17
 * Assignment:  cView-P3
 * Source File: SimpleConvert.cs
 * Language:    C#
 * Course:      CSCI-C 490, C# Programming, MoWe 08:00
 * Purpose:     Provides some simplifified variants of some Convert methods.
-----------------------------------------------------------------------------------------*/

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

namespace Ph3
{
    public static class SimpleConvert
    {
        /*-----------------------------------------------------------------------------------
         * Name:    ToDateTime
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  DateTime object representing either the converted value or the default DateTime.
        -----------------------------------------------------------------------------------*/
        public static DateTime ToDateTime(string value)
        {
            try
            {
                return Convert.ToDateTime(value);
            }
            catch
            {
                return default(DateTime);
            }
        }

        /*-----------------------------------------------------------------------------------
         * Name:    ToInt32
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
         * Output:  int object representing either the converted value or the default int.
        -----------------------------------------------------------------------------------*/
        public static int ToInt32(string value)
        {
            try
            {
                return Convert.ToInt32(value);
            }
            catch
            {
                return default(int);
            }
        }

        /*-----------------------------------------------------------------------------------
         * Name:    ToSingle
         * Type:    Method
         * Purpose: Attempts to convert the given parameter, but returns the default object value if
         *          it fails for any reason.
         * Input:   string value, containing the value on which conversion will be attempted.
```

```
         * Output:  float object representing either the converted value or the default float.
        ----------------------------------------------------------------------------------------*/
        public static float ToSingle(string value)
        {
            try
            {
                return Convert.ToSingle(value);
            }
            catch
            {
                return default(float);
            }
        }
    }
}

```

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)

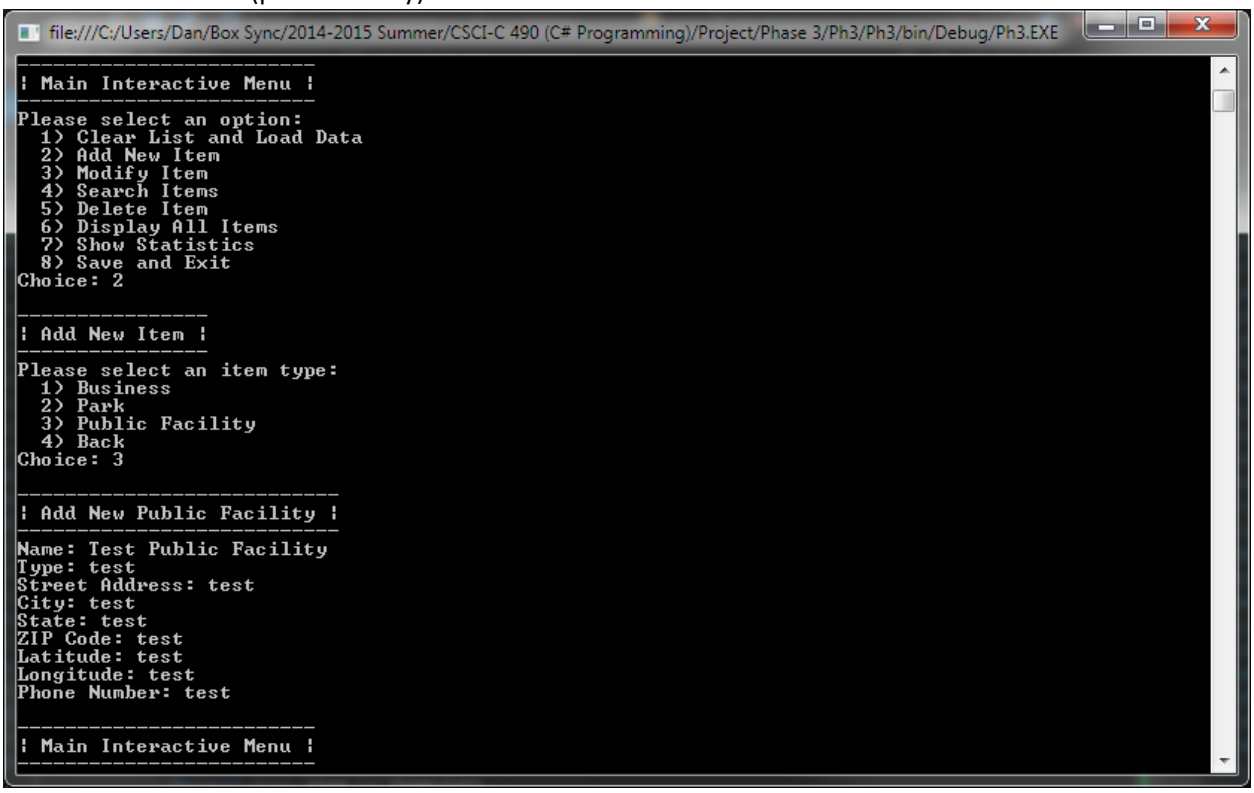
```
file:///C:/Users/Dan/Box Sync/2014-2015 Summer/CSCI-C 490 (C# Programming)/Project/Phase 3/Ph3/Ph3/bin/Debug/Ph3.EXE
-------------------------
| Main Interactive Menu |
-------------------------
Please select an option:
  1) Clear List and Load Data
  2) Add New Item
  3) Modify Item
  4) Search 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)



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, -86.3075075099994)
        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...
```