```
/*-----------------------------------------------------------------------------------------------------
 * 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;
        }
    }
}
```