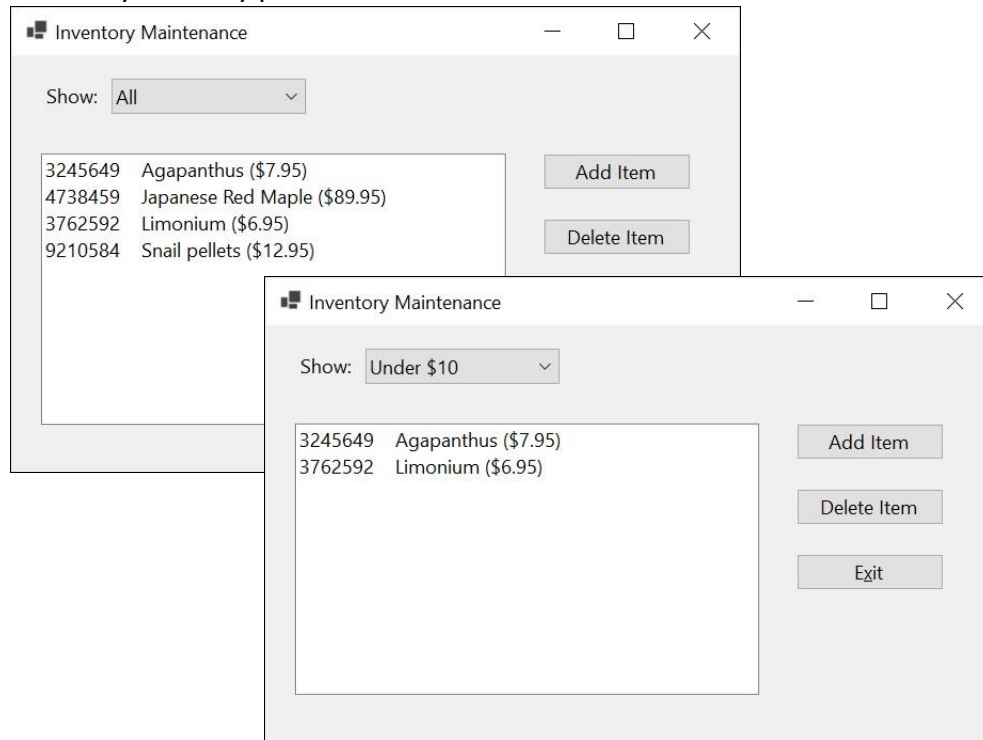


Exercise 2-1**Use LINQ**

In this exercise, you'll use LINQ to update the Inventory Maintenance application so it displays the inventory items in alphabetical order by description, and so it can filter the inventory items by price.

**Review the starting code**

1. Extract the provided start project (Homework 2 Project Starts.zip), and open the project in the "InventoryMaintenance" folder.
2. Display the code for the `InvItemList` class and note that it contains a readonly property named `DisplayText`.
3. Display the code for the **Inventory Maintenance** form and review the code that loads the items in the combo box and the code that executes when the selected item in the combo box is changed.
4. Run the application and view the items in the combo box. Notice that nothing happens when you change the selected item. Exit the application.

Add code that sorts and filters the inventory items

5. Find the `FillItemListBox()` method in the **Inventory Maintenance** form. It starts by storing the selected value of the combo box in a variable

named filter and declaring a local collection of `InvItem` objects named `filteredItems`.

Note: This local collection is of the `IEnumerable<InvItem>` type. That's because LINQ queries return `IEnumerable<T>` collections (see figure 182 in the textbook).

6. Code an if/else statement that uses LINQ to query the class-level collection named `invItems` based on the value of the filter variable. The LINQ queries should also order the inventory items by Description. Assign the result of each LINQ query to the `filteredItems` collection. Note: You can use method-based queries or query expressions here.
7. Update the foreach statement so it loops through the local `filteredItems` collection rather than the class-level `invItems` collection.
8. Test the application to be sure it displays the items in alphabetical order and filters the items by price when the selected item in the combo box changes.
9. Add the following new item to the inventory:

ItemNo: 4372639 Description: Creeping Phlox Price: 24.99

Notice that the new item is sorted so it appears alphabetically in the display. Use the combo box to show items whose price is between \$10 and \$50, and see that the new item is displayed. When you're done, exit the application.
10. Open the `InventoryItems.txt` file and see that the items aren't in alphabetical order, and that the item you just added is the last item in the file.

Add code that uses LINQ to find the inventory item to delete

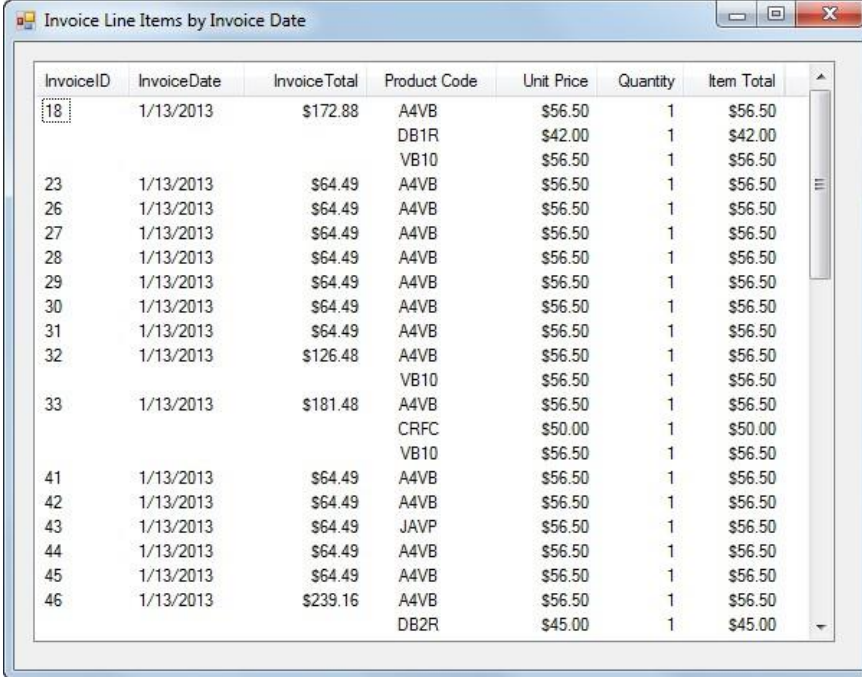
11. Run the application, select the item you added in step 9, and click Delete. Note that the confirmation message identifies the wrong item for deletion. Click Cancel and exit the application.
12. Display the `btnDelete_Click()` event handler in the **Inventory Maintenance** form and comment out the line of code that uses the selected index value of the `ListView` control to get the selected `InvItem` object from the `invItems` collection.
13. Add code that uses that selected index value to retrieve the display text of the selected item from the `Items` collection of the `ListView` control.
14. Use a LINQ query to get the selected `InvItem` object from the `InvItems` collection. The LINQ query should select the item whose `DisplayText` property is equal to the display text value you just retrieved.

Note: To do this, you need to use the `First()` or `FirstOrDefault()` method. Because of that, you should code this as a method-based query.

15. Run the application and try to delete the item you added in step 9. This time, the confirm message should present the correct inventory item. Click OK to delete the item and exit the application.

Exercise 2-2 Use LINQ to create an Invoice Items application

In this exercise, you'll use LINQ to join the data in two `List<>` objects and then display that data in a `ListView` control.



InvoiceID	InvoiceDate	InvoiceTotal	Product Code	Unit Price	Quantity	Item Total
18	1/13/2013	\$172.88	A4VB	\$56.50	1	\$56.50
			DB1R	\$42.00	1	\$42.00
			VB10	\$56.50	1	\$56.50
23	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
26	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
27	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
28	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
29	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
30	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
31	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
32	1/13/2013	\$126.48	A4VB	\$56.50	1	\$56.50
			VB10	\$56.50	1	\$56.50
33	1/13/2013	\$181.48	A4VB	\$56.50	1	\$56.50
			CRFC	\$50.00	1	\$50.00
			VB10	\$56.50	1	\$56.50
41	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
42	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
43	1/13/2013	\$64.49	JAVP	\$56.50	1	\$56.50
44	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
45	1/13/2013	\$64.49	A4VB	\$56.50	1	\$56.50
46	1/13/2013	\$239.16	A4VB	\$56.50	1	\$56.50
			DB2R	\$45.00	1	\$45.00

Design the form

1. The provided start project (Homework 2 Project Starts.zip) which you extracted in Exercise 10-1 above, contains another folder named "InvoiceLineItems". Open the project in that folder. This project contains the **Invoice Line Items** form, along with the business and database classes and database files needed by the application.
2. Add a `ListView` control to the form, and set the View property of this control to Details.
3. Use the smart tag menu for the `ListView` control to display the ColumnHeader Collection Editor. Then, define the column headers for this control so they appear like the first and the last four shown above.

Add code to display the line item data

4. Add an event handler for the Load event of the form. Then, use the `GetLineItems` method in the `LineItemDB` class to get a `List<LineItem>` object, and store this list in a variable.

5. Define a query expression that returns all the line items from the line item list. The select clause for this query expression should select entire line items.
6. Use a foreach statement to execute the query and load the results into the ListView control.
7. Test the application to be sure it displays the line items correctly.

Enhance the application to include invoice data

8. Add two more columns to the ListView control for displaying the invoice date and invoice total. (To get these columns to display before the last four columns, you'll need to set their DisplayIndex properties to 1 and 2.) Then, add a statement to the Load event handler of the form that uses the GetInvoices method of the InvoiceDB class to get a List<Invoice> object, and store the list in a variable.
9. Modify the query expression so it joins the data in the invoice list with the data in the line item list, so it sorts the results by invoice date, and so only the fields that are needed by the form are returned by the query.
10. Modify the foreach statement so it adds the invoice ID, invoice date, and invoice total to the ListView control.
11. Test the application to be sure it displays the invoice data correctly.

Enhance the application so it doesn't repeat invoice information

12. Declare a variable outside the foreach statement to hold an invoice ID, and initialize this variable to 0. Then, add code within the foreach statement that checks if the invoice ID for the current item is equal to the invoice ID in the variable you just declared. If they aren't equal, the invoice ID, invoice date, and invoice total should be added to the ListView control, and the invoice ID variable should be set to the value of the invoice ID for the current item. Otherwise, spaces should be added to the ListView control for these fields.
13. Test this change to be sure it works correctly.