Using Collections

In this chapter you will examine various types of collections such as stacks, queues, dictionaries, hash tables, sorted lists, and array lists. A collection may consist of a group of individual data items or a collection of objects.

Objectives

- Declare and reference collections
- Determine the proper collection type for a specific situation
- Understand various data structures
- Create a collection of objects and add and remove items in the collection
- Add objects to the Items collection of a list box, control the display, and retrieve select objects
- Understand generics and be able to create generic procedures
- Create a generic collection of objects
Referencing Collection Items

A collection may consist of a group of individual items or a group of objects.

- Reference items by either a key (a string value) or an index (the numeric position).
  - Example: Reference the Stores table (first table) in the Tables collection
    - `PubsDataSet.Tables("Stores")`
    - `PubsDataSet.Tables(0)`
    - `PubsDataSet.Tables!Stores`

The words **collection**, **list**, and **data structure** can be used interchangeably.

System.Collections Namespace

- The **System.Collections** namespace provides classes for several different types of collections.
- Some types of collections are based on the way items are handled.
  - A **queue** is like a line: the first item in should be the first one out (**FIFO**)
  - In a **stack** collection, the last one in is the first one out (**LIFO**)
  - A **dictionary** collection consists of a key and value pair
    - `Hashtable`
    - `SortedList`
- All collection classes inherit from the same base class and share many properties and methods.

See table in text on p. 445 for Collection Classes.

Collection Classes Example

The user can choose to create a stack, queue, sorted list, or **hash table** and add and remove items in the collections.
**Using a Stack**
- Provides an unsorted list
- LIFO
- Use the Push method to add an item to the list.
- Use the Pop method to remove an item from the list.
- Use the Peek method to look at the last item without removing it.
- The elements in a stack appear in reverse order from original entry.

**Using a Queue**
- An unsorted list
- First item added is the first one to be removed
- Use the Enqueue method to add items to the queue.
- Use the Dequeue method to remove items from the queue.
- Use Peek method to view the last item.
- Elements in a queue appear in the same order as the original entry.

**Using a Hash Table**
- A dictionary-type collection of key/value pairs
- Fastest type of list for searching
- Keys are calculated using an algorithm that produces a unique key that is used with every calculation.
- Use the GetHashCode method to calculate the key.
  - Calculate the hash code for the key.
    ```csharp
    keyString = itemString.GetHashCode.ToString();
    aHashtable.Add(keyString, itemString);
    ```
  - Calculate the hash code of a selected item.
    ```csharp
    itemString = languagesList.SelectedItem.ToString();
    keyString = itemString.GetHashCode.ToString();
    aHashtable.Remove(keyString);
    ```
Using Hash Tables
(2 of 2)
- Use Add and Remove methods to place/remove items in the list.
- Don’t use a hash table when:
  - Data must be in a specific order
  - There may be duplicate keys
- Refer to the collection of items in a hash table using the Values property

Sorted Lists
- A list of key and value pairs automatically sorted by the key
- Use the Add, Remove, and RemoveAt methods
- Access elements by key, by value, or by index
- Keys must be unique and may be created from a hash code calculation
  ```csharp
  String keyString = ItemString.Substring(0, 3)
  sSortedList.Add(keyString, ItemString)
  ```

Using Array Lists
- Dynamically increase in size as new elements are added
- Use the Capacity property to set the size of the list
- Capacity is automatically increased in chunks as elements are added
- Use the TrimToSize method to reduce the size of the collection
  ```csharp
  See the Array List Properties and Methods p. 452
  ```
Creating a Collection of Objects

- Create a collection class and add objects to the collection
- Choose the collection class type based on needs for speed, sorting, retrieval by value, index, or key
- Refer to the member of a collection in two ways:
  - Specify an index number if order does not change
  - Give each object a unique string key (PIN, customer number, account number)
  - Can use a hashing algorithm to create

A Collection of Student Objects

- Example: Create and access a collection of student objects
- Add or remove students from a collection, or display the GPA for a selected student

Declaring a Collection

- Declare and instantiate the collection at the class level.
- Select the best collection type for the application

Example: using a sorted list and a hash code of the name as the key

Declare class level variable.
Private studentsHashtable as Hashtable
Adding Objects to a Collection

- Create the key using the GetHashCode method
  - Performs a calculation on the name and produces an integer
- Convert the key to a string
- Add the object to the collection

```
Dim aStudent As New Student(nameTextBox.Text, _
  Decimal.Parse(gpaTextBox.Text))
keyString = aStudent.Name.GetHashCode.ToString
studentsHashtable.Add(keyString, aStudent)
```

Removing an Element from a Collection

- Remove method deletes an element by key (string)
- RemoveAt removes an element by index (numeric)
- If keyless, such an ArrayList
- Remove method removes a specific object
- RemoveAt deletes by index

```
Dim keyString As String = studentsListBox.SelectedItem. GetHashCode.ToString
studentsHashtable.Remove(keyString)
```

Retrieving an Element from a Collection

- Use the Item property to retrieve an object from a list
- For most collection types, the Item property is defined as the default property; the word "Item" can be left out of statements

```
Dim keyString As String = studentsListBox.SelectedItem. GetHashCode.ToString
Dim aStudent As Student = CType(studentsHashtable.Item(keyString), Student())
```
Using For Each / Next

- Use a For Each/Next structure to access each object in a collection
- Dictionary-type collections (sorted lists and hash tables) return an object of DictionaryEntry data type
  - Cast the element to the object type needed

Using an Items Collection

- The Items collection of a list box or combo box is a collection of objects; not just strings
- Add objects to the Items collection and retrieve the complete object in the Item property
- If necessary, write a ToString method in the object’s class to override the base class

```vbnet
Public Overrides Function ToString() As String
    Return nameString
End Function
```

Generics

- Great advantage of object-oriented programming is the ability to reuse created objects
- Objects created must be generic enough to be used in many programming situations
- Generics provide a way to define strongly typed procedures or classes where the object type is not declared until the user uses the procedure or instantiates the class
  - Don’t need to specify what type of variable object is going to be used until it is actually used
Generic Classes

- Allows users to instantiate strongly type classes while maintaining a high degree of reusability
- After creating a new class, instantiate it and then call the method of the class
- VS 2005 IDE is smart enough to know that if a class is instantiated with a type of Integer, coding the method call prompts for integers

Generic Collections

- In VS 2005 generic “wrappers” can be written around a class which guarantees that all objects in the collection are the same type (string, class object, integer)
- Provide a way to define strongly typed collections where the collection type is not declared until the user instantiates the class
  - Can be used with any object type
  - Instantiate the generic collection class, specifying the needed type and from then on that collection is “locked in” to the specified type
- VS 2005 includes a number of generic collections in the Systems.Collections.Generic namespace