27

Collections

© 2006 Pearson Education, Inc. All rights reserved.

The shapes a bright container can contain!

— Theodore Roethke

Not by age but by capacity is wisdom acquired.

— Titus Maccius Plautus

It is a riddle wrapped in a mystery inside an enigma.

— Winston Churchill

I think this is the most extraordinary collection of talent, of human knowledge, that has ever been gathered together at the White House—with the possible exception of when Thomas Jefferson dined alone.

— John F. Kennedy

© 2006 Pearson Education, Inc. All rights reserved.

OBJECTIVES

In this chapter you will learn:

- The non-generic and generic collections that are provided by the .NET Framework.
- To use class `Array`'s `static` methods to manipulate arrays.
- To use enumerators to "walk through" a collection.
- To use the `foreach` statement with the .NET collections.
- To use non-generic collection classes `ArrayList`, `Stack`, and `Hashtable`.
- To use generic collection classes `SortedDictionary` and `LinkedList`.
- To use synchronization wrappers to make collections safe in multithreaded applications.

© 2006 Pearson Education, Inc. All rights reserved.

Outline

27.1 Introduction
27.2 Collections Overview
27.3 Class `Array` and Enumerators
27.4 Non-Generic Collections
- 27.4.1 Class `ArrayList`
- 27.4.2 Class `Stack`
- 27.4.3 Class `Hashtable`

27.5 Generic Collections
- 27.5.1 Generic Class `SortedDictionary`
- 27.5.2 Generic Class `LinkedList`

27.6 Synchronized Collections
27.7 Wrap-Up

© 2006 Pearson Education, Inc. All rights reserved.

27.1 Introduction

- .NET Framework Collections
  - Prepackaged data-structure classes
    - Known as collection classes
    - Store collections of data
  - Collection classes
    - Enable programmers to store sets of items by using existing data structures
      - Need not concern how they are implemented
    - Example of code reuse
    - Programmers can code faster and expect excellent performance, maximizing execution speed and minimizing memory consumption
  - The .NET Framework provides three namespaces dedicated to collections:
    - `System.Collections` namespace
      - Contains collections that store references to objects
    - `System.Collections.Generic` namespace
      - Contains generic classes to store collections of specified types
    - `System.Collections.Specialized` namespace
      - Contains several collections that support specific types
        - Ex: strings and bits

© 2006 Pearson Education, Inc. All rights reserved.

27.2 Collections Overview

- **Collections Overview**
  - **Implements some combination of the collection interfaces**
    - **Declare the operations to be performed generically on various types of collections**
  - **Store any object in a collection**
  - **Retrieving object references from a collection**
    - **Object references obtained from a collection typically need to be downcast to an appropriate type**
      - **Allow the application to process the objects correctly**
  - **Many of these new classes from `System.Collections.Generic` namespace are simply generic counterparts of the classes in namespace `System.Collections`**
    - **Specify the type stored in a collection and any item retrieved from the collection will have the correct type**
      - **Eliminates the need for explicit type casts**
      - **Eliminates the overhead of explicit casting**
        - **Improve efficiency**

© 2006 Pearson Education, Inc. All rights reserved.

Interface	Description
`ICollection`	The root interface in the collections hierarchy from which interfaces `IList` and `IDictionary` inherit. Contains a `Count` property to determine the size of a collection and a `CopyTo` method for copying a collection's contents into a traditional array.
`IList`	An ordered collection that can be manipulated like an array. Provides an indexer for accessing elements with an `int` index. Also has methods for searching and modifying a collection, including `Add`, `Remove`, `Contains` and `IndexOf`.
`IDictionary`	A collection of values, indexed by an arbitrary "key" object. Provides an indexer for accessing elements with an `object` index and methods for modifying the collection (e.g. `Add`, `Remove`). `IDictionary` property `Keys` contains the `object`s used as indices, and property `Values` contains all the stored `object`s.
`IEnumerable`	An object that can be enumerated. This interface contains exactly one method, `GetEnumerator`, which returns an `IEnumerator` object (discussed in Section 27.3). `ICollection` implements `IEnumerable`, so all collection classes implement `IEnumerable` directly or indirectly.

Fig. 27.1 | Some common collection interfaces.

© 2006 Pearson Education, Inc. All rights reserved.

Class	Implements	Description
`System` *namespace:*		
`Array`	`IList`	The base class of all conventional arrays. See Section 27.3.
`System.Collections` *namespace:*		
`ArrayList`	`IList`	Mimics conventional arrays, but will grow or shrink as needed to accommodate the number of elements. See Section 27.4.1.
`BitArray`	`ICollection`	A memory-efficient array of `bool`s.
`Hashtable`	`IDictionary`	An unordered collection of key–value pairs that can be accessed by key. See Section 27.4.3.
`Queue`	`ICollection`	A first-in first-out collection. See Section 25.6.
`SortedList`	`IDictionary`	A generic `Hashtable` that sorts data by keys and can be accessed either by key or by index.
`Stack`	`ICollection`	A last-in, first-out collection. See Section 27.4.2.

Fig. 27.2 | Some collection classes of the .NET Framework. (Part 1 of 2.)

© 2006 Pearson Education, Inc. All rights reserved.

Class	Implements	Description
System.Collections.Generic namespace:		
Dictionary< K, E >	IDictionary< K, E >	A generic, unordered collection of key–value pairs that can be accessed by key.
LinkedList< E >	ICollection< E >	A doubly linked list. See Section 27.5.2.
List< E >	IList< E >	A generic ArrayList.
Queue< E >	ICollection< E >	A generic Queue.
SortedDictionary< K, E >	IDictionary< K, E >	A Dictionary that sorts the data by the keys in a binary tree. See Section 27.5.1.
SortedList< K, E >	IDictionary< K, E >	A generic SortedList.
Stack< E >	ICollection< E >	A generic Stack.
[*Note: All collection classes directly or indirectly implement ICollection and IEnumerable (or the equivalent generic interfaces ICollection< E > and IEnumerable< E > for generic collections).*]		

Fig. 27.2 | Some collection classes of the .NET Framework. (Part 2 of 2.)

© 2006 Pearson Education, Inc. All rights reserved.

27.3 Class `Array` and Enumerators

- **Class `Array`**
  - **All arrays implicitly inherit from this abstract base class**
    - **Defines property `Length`**
      - **Specifies the number of elements in the array**
    - **Provides `static` methods that provide algorithms for processing arrays**
      - **For a complete list of class `Array`’s methods visit:**

        msdn2.microsoft.com/en-us/library/system.array.aspx

© 2006 Pearson Education, Inc. All rights reserved.

Outline

UsingArray.cs

(1 of 4)

```
1  // Fig. 27.3: UsingArray.cs
2  // Array class static methods for common array manipulations.
3  using System;
4  using System.Collections;
5
6  // demonstrate algorithms of class Array
7  public class UsingArray
8  {
9     private static int[] intValues = { 1, 2, 3, 4, 5, 6 };
10    private static double[] doubleValues = { 8.4, 9.3, 0.2, 7.9, 3.4 };
11    private static int[] intValuesCopy;
12
13    // method Main demonstrates class Array's methods
14    public static void Main( string[] args )
15    {
16       intValuesCopy = new int[ intValues.Length ]; // defaults to zeroes
17
18       Console.WriteLine( "Initial array values:\n" );
19       PrintArrays(); // output initial array contents
20
21       // sort doubleValues
22       Array.Sort( doubleValues );
23
24       // copy intValues into intValuesCopy
25       Array.Copy( intValues, intValuesCopy, intValues.Length );
26
27       Console.WriteLine( "\nArray values after Sort and Copy:\n" );
28       PrintArrays(); // output array contents
29       Console.WriteLine();
```

Declare three `static` array variables

Initialize `intValuesCopy` as the same size as `intValues`

Sort the array `doubleValues` in ascending order

Copy elements from array `intValues` to array `intValuesCopy`

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
30
31       // search for 5 in intValues
32       int result = Array.BinarySearch( intValues, 5 );
33       if ( result >= 0 )
34          Console.WriteLine( "5 found at element {0} in intValues",
35             result );
36       else
37          Console.WriteLine( "5 not found in intValues" );
38
39       // search for 8763 in intValues
40       result = Array.BinarySearch( intValues, 8763 );
41       if ( result >= 0 )
42          Console.WriteLine( "8763 found at element {0} in intValues",
43             result );
44       else
45          Console.WriteLine( "8763 not found in intValues" );
46    } // end method Main
```

UsingArray.cs

Perform binary searches on array `intValues`

© 2006 Pearson Education, Inc. All rights reserved.

Outline

UsingArray.cs

```
47
48    // output array content with enumerators
49    private static void PrintArrays()
50    {
51       Console.Write( "doubleValues: " );
52
53       // iterate through the double array with an enumerator
54       IEnumerator enumerator = doubleValues.GetEnumerator();
55
56       while ( enumerator.MoveNext() )
57          Console.Write( enumerator.Current + " " );
58
59       Console.Write( "\nintValues: " );
60
61       // iterate through the int array with an enumerator
62       enumerator = intValues.GetEnumerator();
63
64       while ( enumerator.MoveNext() )
65          Console.Write( enumerator.Current + " " );
66
67       Console.Write( "\nintValuesCopy: " );
68
69       // iterate through the second int array with a foreach statement
70       foreach ( int element in intValuesCopy )
71          Console.Write( element + " " );
72
73       Console.WriteLine();
74    } // end method PrintArrays
75 } // end class UsingArray
```

Obtains an enumerator for the corresponding array

Advances the enumerator for the corresponding array

Obtains and output the current array element

Iterate over the collection elements like an enumerator

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
Initial array values:

doubleValues: 8.4 9.3 0.2 7.9 3.4
intValues: 1 2 3 4 5 6
intValuesCopy: 0 0 0 0 0 0

Array values after Sort and Copy:

doubleValues: 0.2 3.4 7.9 8.4 9.3
intValues: 1 2 3 4 5 6
intValuesCopy: 1 2 3 4 5 6

5 found at element 4 in intValues
8763 not found in intValues
```

`UsingArray.cs`

(4 of 4)

© 2006 Pearson Education, Inc. All rights reserved.

27.3 Class `Array` and Enumerators (Cont.)

- `Array` **Methods**
  - `Sort`
    - Sort array
    - Returns the array containing its original elements sorted in ascending order
  - `Copy`
    - Copy elements from an array to another
    - 1st argument is the array to copy
    - 2nd argument is the destination array
    - 3rd argument is an `int` representing the number of elements to copy
  - `BinarySearch`
    - Perform binary searches on array
    - Receives a *sorted* array in which to search and the key for which to search
    - Returns the index in the array at which it finds the key
  - `Clear`
    - Set a range of elements to 0 or `null`
  - `CreateInstance`
    - Create a new array of a specified type
  - `IndexOf`
    - Locate the first occurrence of an object in an array or portion of an array
  - `LastIndexOf`
    - Locate the last occurrence of an object in an array or portion of an array
  - `Reverse`
    - Reverse the contents of an array or portion of an array

© 2006 Pearson Education, Inc. All rights reserved.

Common Programming Error 27.1

Passing an unsorted array to `BinarySearch` is a logic error—the value returned is undefined.

© 2006 Pearson Education, Inc. All rights reserved.

27.3 Class `Array` and Enumerators (Cont.)

- **Enumerators**
  - `GetEnumerator` **method**
    - **Obtains an enumerator for an array**
  - `Array` **implements the** `IEnumerable` **interface**
  - **Interface** `IEnumerator`:
    - **Method** `MoveNext`
      - **Moves the enumerator to the next element in the collection**
      - **Returns** `true` **if there is at least one more element in the collection**
    - **Method** `Reset`
      - **Positions the enumerator before the first element of the collection**
    - **Method** `Current`
      - **Returns the object at the current location in the collection**
    - **Note on methods** `MoveNext` **and** `Reset`
      - **Throws an** `InvalidOperationException`
        - **If the contents of the collection are modified in any way after the enumerator is created**

© 2006 Pearson Education, Inc. All rights reserved.

27.3 Class `Array` and Enumerators (Cont.)

- `foreach` **Statement**
  - **Implicitly obtains an enumerator**
    - **Via the** `GetEnumerator` **method**
  - **Uses the enumerator's** `MoveNext` **method and** `Current` **property to traverse the collection**
  - **Able to iterate over *any* collection that implements the** `IEnumerable` **interface**

© 2006 Pearson Education, Inc. All rights reserved.

Common Programming Error 27.2

If a collection is modified after an enumerator is created for that collection, the enumerator immediately becomes invalid—any methods called with the enumerator after this point throw `InvalidOperationException`s. For this reason, enumerators are said to be "fail fast."

© 2006 Pearson Education, Inc. All rights reserved.

27.4 Non-Generic Collections

- Class `ArrayList`
  - Mimics the functionality of conventional arrays
  - Provides dynamic resizing of the collection through the class's methods
    - Property `Capacity`
      - Manipulate the capacity of the `ArrayList`
      - When `ArrayList` needs to grow, it by default doubles its `Capacity`
  - Store references to objects
    - All classes derive from class `object`
      - Can contain objects of any type

© 2006 Pearson Education, Inc. All rights reserved.

Performance Tip 27.1

As with linked lists, inserting additional elements into an `ArrayList` whose current size is less than its capacity is a fast operation.

© 2006 Pearson Education, Inc. All rights reserved.

Performance Tip 27.2

It is a slow operation to insert an element into an `ArrayList` that needs to grow larger to accommodate a new element. An `ArrayList` that is at its capacity must have its memory reallocated and the existing values copied into it.

© 2006 Pearson Education, Inc. All rights reserved.

Performance Tip 27.3

If storage is at a premium, use method `TrimToSize` of class `ArrayList` to trim an `ArrayList` to its exact size. This will optimize an `ArrayList`'s memory use. Be careful—if the application needs to insert additional elements, the process will be slower because the `ArrayList` must grow dynamically (trimming leaves no room for growth).

© 2006 Pearson Education, Inc. All rights reserved.

Performance Tip 27.4

The default capacity increment, doubling the size of the `ArrayList`, may seem to waste storage, but doubling is an efficient way for an `ArrayList` to grow quickly to "about the right size." This is a much more efficient use of time than growing the `ArrayList` by one element at a time in response to insert operations.

© 2006 Pearson Education, Inc. All rights reserved.

Method or Property	Description
`Add`	Adds an `object` to the `ArrayList` and returns an `int` specifying the index at which the `object` was added.
`Capacity`	Property that gets and sets the number of elements for which space is currently reserved in the `ArrayList`.
`Clear`	Removes all the elements from the `ArrayList`.
`Contains`	Returns `true` if the specified `object` is in the `ArrayList`; otherwise, returns `false`.
`Count`	Read-only property that gets the number of elements stored in the `ArrayList`.
`IndexOf`	Returns the index of the first occurrence of the specified `object` in the `ArrayList`.
`Insert`	Inserts an `object` at the specified index.
`Remove`	Removes the first occurrence of the specified `object`.
`RemoveAt`	Removes an object at the specified index.
`RemoveRange`	Removes a specified number of elements starting at a specified index in the `ArrayList`.
`Sort`	Sorts the `ArrayList`.
`TrimToSize`	Sets the `Capacity` of the `ArrayList` to the number of elements the `ArrayList` currently contains (`Count`).

Fig. 27.4 | Some methods and properties of class `ArrayList`.

© 2006 Pearson Education, Inc. All rights reserved.

Outline

ArrayListTest.cs

Declare two arrays of `string`s

Create an `ArrayList` with an initial capacity of one element

Add the five elements of array `colors` to `list`

Use overloaded constructor to create a new `ArrayList` initialized with the contents of array `removeColors`

© 2006 Pearson Education, Inc. All rights reserved.

Outline

`ArrayListTest.cs`

(2 of 3)

Iterate through `arrayList` to output its elements

Display the current number of elements and the maximum number of elements that can be stored without allocating more memory

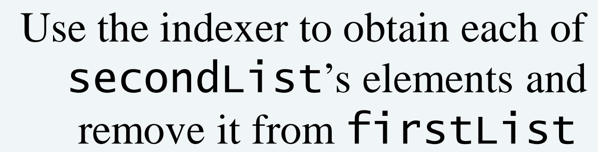
Determine the position of the `string` “Blue” in `arrayList`

© 2006 Pearson Education, Inc. All rights reserved.

Outline

ArrayListTest.cs

(3 of 3)



© 2006 Pearson Education, Inc. All rights reserved.

Performance Tip 27.5

`ArrayList` methods `IndexOf` and `Contains` each perform a linear search, which is a costly operation for large `ArrayList`s. If the `ArrayList` is sorted, use `ArrayList` method `BinarySearch` to perform a more efficient search. Method `BinarySearch` returns the index of the element, or a negative number if the element is not found.

© 2006 Pearson Education, Inc. All rights reserved.

27.4 Non-Generic Collections (Cont.)

- Class `Stack`
  - Contains methods `Push` and `Pop` to perform the basic stack operations
  - Method `Push`
    - Takes and inserts an object to the top of the `Stack`
    - Grows to accommodate more objects
      - When the number of items on the `Stack` (the `Count` property) is equal to the capacity at the time of the `Push` operation
    - Can store only references to objects
      - Value types are implicitly boxed before they are added
  - Method `Pop`
    - Removes and returns the object current on the top of the `Stack`
  - Method `Peek`
    - Returns the value of the top stack element
    - Does not remove the element from the `Stack`
  - Note on methods `Pop` and `Peek`
    - Throws `InvalidOperationException` if the `Stack` is empty
  - Property `Count`
    - Obtain the number of elements in `Stack`

© 2006 Pearson Education, Inc. All rights reserved.

Outline

StackTest.cs

(1 of 3)

```
1  // Fig. 27.6: StackTest.cs
2  // Demonstrating class Stack.
3  using System;
4  using System.Collections;
5
6  public class StackTest
7  {
8     public static void Main( string[] args )
9     {
10       Stack stack = new Stack(); // default Capacity of 10
11
12       // create objects to store in the stack
13       bool aBoolean = true;
14       char aCharacter = '$';
15       int anInteger = 34567;
16       string aString = "hello";
17
18       // use method Push to add items to (the top of) the stack
19       stack.Push( aBoolean );
20       PrintStack( stack );
21       stack.Push( aCharacter );
22       PrintStack( stack );
23       stack.Push( anInteger );
24       PrintStack( stack );
25       stack.Push( aString );
26       PrintStack( stack );
27
28       // check the top element of the stack
29       Console.WriteLine( "The top element of the stack is {0}\n",
30          stack.Peek() );
```

Create a stack with the default initial capacity of 10 elements

Add four elements to the stack

Obtain the value of the top stack element

© 2006 Pearson Education, Inc. All rights reserved.

Outline

StackTest.cs


```
31
32       // remove items from stack
33       try
34       {
35          while ( true )
36          {
37             object removedObject = stack.Pop();
38             Console.WriteLine( removedObject + " popped" );
39             PrintStack( stack );
40          } // end while
41       } // end try
42       catch ( InvalidOperationException exception )
43       {
44          // if exception occurs, print stack trace
45          Console.Error.WriteLine( exception );
46       } // end catch
47    } // end Main
48
49    // print the contents of a stack
50    private static void PrintStack( Stack stack )
51    {
52       if ( stack.Count == 0 )
53          Console.WriteLine( "stack is empty\n" ); // the stack is empty
54       else
55       {
56          Console.Write( "The stack is: " );
57
58          // iterate through the stack with a foreach statement
59          foreach ( object element in stack )
60             Console.Write( "{0} ", element ); // invokes ToString
```

Obtain and remove the value of the top stack element

Use `foreach` statement to iterate over the stack and output its contents

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
61
62          Console.WriteLine( "\n" );
63       } // end else
64    } // end method PrintStack
65 } // end class StackTest
```

```
The stack is: True

The stack is: $ True

The stack is: 34567 $ True

The stack is: hello 34567 $ True

The top element of the stack is hello

hello popped
The stack is: 34567 $ True

34567 popped
The stack is: $ True

$ popped
The stack is: True

True popped
stack is empty

System.InvalidOperationException: Stack empty.
   at System.Collections.Stack.Pop()
   at StackTest.Main(String[] args) in C:\examples\ch27\
      fig27_06\StackTest\StackTest.cs:line 37
```

`StackTest.cs`

(3 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

Common Programming Error 27.3

Attempting to `Peek` or `Pop` an empty `Stack` (a `Stack` whose `Count` property is `0`) causes an `InvalidOperationException`.

© 2006 Pearson Education, Inc. All rights reserved.

27.4 Non-Generic Collections (Cont.)

- Class `HashTable`
  - Hashing
    - Convert the application key rapidly to an index and the information could be store/retrieve at that location in the array
    - The data is stored in a data structure called a hash table
    - Convert a key into an array subscript
      - Literally scramble the bits
        - Makes a “hash” of the number

© 2006 Pearson Education, Inc. All rights reserved.

27.4 Non-Generic Collections (Cont.)

- Collisions
  - Problem: Two different keys "hash into" the same cell in the array
    - Solution 1: "Hash Again"
      - Hashing process is designed to be quite random
        - Assumption that within a few hashes, an available cell will be found
    - Solution 2: Uses one hash to locate the first candidate cell. If the cell is occupied, successive cells are searched linearly until an available cell is found
      - Retrieval works the same way
        - The key is hashed once, the resulting cell is checked to determine whether it contains the desired data
          - If it does, the search is complete
          - If it does not, successive cells are searched linearly until the desired data is found
    - Solution 3: Have each cell of the table be a hash "bucket" of all the key–value pairs that hash to that cell
      - Typically as a linked list
      - .NET Framework's `Hashtable` class implements this solution

© 2006 Pearson Education, Inc. All rights reserved.

27.4 Non-Generic Collections (Cont.)

- The load factor
  - The ratio of the number of objects stored in the hash table to the total number of cells of the hash table
  - Affects the performance of hashing schemes
    - The chance of collisions tends to increase as this ratio gets higher
- A hash function
  - Performs a calculation that determines where to place data in the hash table
  - Applied to the key in a key–value pair of objects
- Class `Hashtable` can accept any object as a key
  - Class `object` defines method `GetHashCode` that all objects inherit

© 2006 Pearson Education, Inc. All rights reserved.

Performance Tip 27.6

The load factor in a hash table is a classic example of a space/time trade-off: By increasing the load factor, we get better memory utilization, but the application runs slower due to increased hashing collisions. By decreasing the load factor, we get better application speed because of reduced hashing collisions, but we get poorer memory utilization because a larger portion of the hash table remains empty.

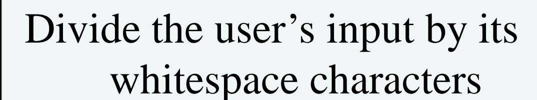
© 2006 Pearson Education, Inc. All rights reserved.

Outline

HashtableTest.cs

(1 of 3)

```
1  // Fig. 27.7: HashtableTest.cs
2  // Application counts the number of occurrences of each word in a string
3  // and stores them in a hash table.
4  using System;
5  using System.Text.RegularExpressions;
6  using System.Collections;
7
8  public class HashtableTest
9  {
10    public static void Main( string[] args )
11    {
12       // create hash table based on user input
13       Hashtable table = CollectWords();
14
15       // display hash table content
16       DisplayHashtable( table );
17    } // end method Main
18
19    // create hash table from user input
20    private static Hashtable CollectWords()
21    {
22       Hashtable table = new Hashtable(); // create a new hash table
23
24       Console.WriteLine( "Enter a string: " ); // prompt for user input
25       string input = Console.ReadLine(); // get input
26
27       // split input text into tokens
28       string[] words = Regex.Split( input, @"\s+" );
```



© 2006 Pearson Education, Inc. All rights reserved.

```
29
30       // processing input words
31       foreach ( string word in words )
32       {
33          string wordKey = word.ToLower(); // get word in lowercase
34
35          // if the hash table contains the word
36          if ( table.ContainsKey( wordKey ) )
37          {
38             table[ wordKey ] = ( ( int ) table[ wordKey ] ) + 1;
39          } // end if
40          else
41             // add new word with a count of 1 to hash table
42             table.Add( wordKey, 1 );
43       } // end foreach
44
45       return table;
46    } // end method CollectWords
47
48    // display hash table content
49    private static void DisplayHashtable( Hashtable table )
50    {
51       Console.WriteLine( "\nHashtable contains:\n{0,-12}{1,-12}",
52          "Key:", "Value:" );
```

Convert each word to lowercase

Determine if the word is in the hash table

Use indexer to obtain and set the key's associated value

Create a new entry in the hash table and set its value to 1

HashtableTest.cs

(2 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
53
54        // generate output for each key in hash table
55        // by iterating through the Keys property with a foreach statement
56        foreach ( object key in table.Keys )
57           Console.WriteLine( "{0,-12}{1,-12}", key, table[ key ] );
58
59        Console.WriteLine( "\nsize: {0}", table.Count );
60     } // end method DisplayHashtable
61  } // end class HashtableTest
```

```
Enter a string:
As idle as a painted ship upon a painted ocean

Hashtable contains:
Key:         Value:
painted      2
a            2
upon         1
as           2
ship         1
idle         1
ocean        1

size: 7
```

HashtableTest.cs

Get an ICollection that contains all the keys

Iterate through the hash table and output its elements

© 2006 Pearson Education, Inc. All rights reserved.

Common Programming Error 27.4

Using the `Add` method to add a key that already exists in the hash table causes an `ArgumentException`.

© 2006 Pearson Education, Inc. All rights reserved.

27.4 Non-Generic Collections (Cont.)

- Class `Hashtable`
  - Method `ContainsKey`
    - Determine whether the word is in the hash table
  - Property `Keys`
    - Get an `ICollection` that contains all the keys in the hash table
  - Property `Value`
    - Gets an `ICollection` of all the values stored in the hash table
  - Property `Count`
    - Get the number of key-value pairs in the hash table

© 2006 Pearson Education, Inc. All rights reserved.

27.4 Non-Generic Collections (Cont.)

- **Problems with Non-Generic Collections**
  - `Hashtable` **stores its keys and data as** `object` **references**
    - **Say we store only** `string` **keys and** `int` **values by convention**
      - **Inefficient!**
  - **Cannot control what is being put into the** `Hashtable`
    - `InvalidCastException` **might be thrown at execution time**

© 2006 Pearson Education, Inc. All rights reserved.

27.5 Generic Collections

- Generic Class `SortedDictionary`
  - Dictionary
    - A general term for a collection of key–value pairs
      - A hash table is one way to implement a dictionary
  - Does not use a hash table
    - Stores its key–value pairs in a binary search tree
      - Entries are sorted in the tree by key
        - Using the `IComparable` interface
      - Use the same `public` methods, properties and indexers with classes `Hashtable` and `SortedDictionary`
      - Takes two type arguments delimited by < >
        - The first specifies the type of key
        - The second specifies the type of value

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
1  // Fig. 27.8: SortedDictionaryTest.cs
2  // Application counts the number of occurrences of each word in a string
3  // and stores them in a generic sorted dictionary.
4  using System;
5  using System.Text.RegularExpressions;
6  using System.Collections.Generic;
7
8  public class SortedDictionaryTest
9  {
10    public static void Main( string[] args )
11    {
12       // create sorted dictionary based on user input
13       SortedDictionary< string, int > dictionary = CollectWords();
14
15       // display sorted dictionary content
16       DisplayDictionary( dictionary );
17    } // end method Main
18
19    // create sorted dictionary from user input
20    private static SortedDictionary< string, int > CollectWords()
21    {
22       // create a new sorted dictionary
23       SortedDictionary< string, int > dictionary =
24          new SortedDictionary< string, int >();
25
26       Console.WriteLine( "Enter a string: " ); // prompt for user input
27       string input = Console.ReadLine(); // get input
28
29       // split input text into tokens
30       string[] words = Regex.Split( input, @"\s+" );
```

Namespace that contains class `SortedDictionary`

Create a dictionary of `int` values keyed with `string`s

Divide the user’s input by its whitespace characters

`SortedDictionaryTest.cs`

(1 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

```
31
32      // processing input words
33      foreach ( string word in words )
34      {
35         string wordKey = word.ToLower(); // get word in lowercase
36
37         // if the dictionary contains the word
38         if ( dictionary.ContainsKey( wordKey ) )
39         {
40            ++dictionary[ wordKey ];
41         } // end if
42         else
43            // add new word with a count of 1 to the dictionary
44            dictionary.Add( wordKey, 1 );
45      } // end foreach
46
47      return dictionary;
48   } // end method CollectWords
49
50   // display dictionary content
51   private static void DisplayDictionary< K, V >(
52      SortedDictionary< K, V > dictionary )
53   {
54      Console.WriteLine( "\nSorted dictionary contains:\n{0,-12}{1,-12}",
55         "Key:", "Value:" );
```

Convert each word to lowercase

Determine if the word is in the dictionary

Use indexer to obtain and set the key's associated value

Create a new entry in the dictionary and set its value to 1

Modified to be completely generic; takes type parameters K and V

SortedDictionaryTest.cs

(2 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

Outline

SortedDictionaryTest.cs

```
56
57       // generate output for each key in the sorted dictionary
58       // by iterating through the Keys property with a foreach statement
59       foreach ( K key in dictionary.Keys )
60          Console.WriteLine( "{0,-12}{1,-12}", key, dictionary[ key ] );
61
62       Console.WriteLine( "\nsize: {0}", dictionary.Count );
63    } // end method DisplayDictionary
64 } // end class SortedDictionaryTest
```

```
Enter a string:
We few, we happy few, we band of brothers

Sorted dictionary contains:
Key:        Value:
band        1
brothers    1
few,        2
happy       1
of          1
we          3

size: 6
```

Get an ICollection that contains all the keys

Output the number of different words

Iterate through the dictionary and output its elements

© 2006 Pearson Education, Inc. All rights reserved.

Performance Tip 27.7

Because class `SortedDictionary` keeps its elements sorted in a binary tree, obtaining or inserting a key–value pair takes O(log n) time, which is fast compared to linear searching then inserting.

© 2006 Pearson Education, Inc. All rights reserved.

Common Programming Error 27.5

Invoking the `get` accessor of a `SortedDictionary` indexer with a key that does not exist in the collection causes a `KeyNotFoundException`. This behavior is different from that of the `Hashtable` indexer's `get` accessor, which would return `null`.

© 2006 Pearson Education, Inc. All rights reserved.

27.5 Generic Collections (Cont.)

- **Generic Class `LinkedList`**
  - **Doubly-linked list**
  - **Each node contains:**
    - **Property `Value`**
      - **Matches `LinkedList`'s single type parameter**
        - **Contains the data stored in the node**
    - **Read-only property `Previous`**
      - **Gets a reference to the preceding node (or `null` if the node is the first of the list)**
    - **Read-only property `Next`**
      - **Gets a reference to the subsequent reference (or `null` if the node is the last of the list)**
  - **Method `AddLast`**
    - **Creates a new `LinkedListNode`**
    - **Appends this node to the end of the list**
  - **Method `AddFirst`**
    - **Inserts a node at the beginning of the list**
  - **Method `Find`**
    - **Performs a linear search on the list**
    - **Returns the first node that contains a value equal to the passed argument**
      - **Returns `null` if the value is not found**
  - **Method `Remove`**
    - **Splices that node out of the `LinkedList`**
    - **Fixes the references of the surrounding nodes**
  - **One `LinkedListNode` cannot be a member of more than one `LinkedList`**
    - **Generates an `InvalidOperationException`**

© 2006 Pearson Education, Inc. All rights reserved.

Outline

LinkedListTest.cs

```
1  // Fig. 27.9: LinkedListTest.cs
2  // Using LinkedLists.
3  using System;
4  using System.Collections.Generic;
5
6  public class LinkedListTest
7  {
8     private static readonly string[] colors = { "black", "yellow",
9        "green", "blue", "violet", "silver" };
10    private static readonly string[] colors2 = { "gold", "white",
11       "brown", "blue", "gray" };
12
13    // set up and manipulate LinkedList objects
14    public static void Main( string[] args )
15    {
16       LinkedList< string > list1 = new LinkedList< string >();
17
18       // add elements to first linked list
19       foreach ( string color in colors )
20          list1.AddLast( color );
```

Declare two arrays of `strings`

Create a generic `LinkedList` of type `string`

Create and append nodes of array `color`’s elements to the end of the linked list

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
21
22       // add elements to second linked list via constructor
23       LinkedList< string > list2 = new LinkedList< string >( colors2 );
24
25       Concatenate( list1, list2 ); // concatenate list2 onto list1
26       PrintList( list1 ); // print list1 elements
27
28       Console.WriteLine( "\nConverting strings in list1 to uppercase\n" );
29       ToUppercaseStrings( list1 ); // convert to uppercase string
30       PrintList( list1 ); // print list1 elements
31
32       Console.WriteLine( "\nDeleting strings between BLACK and BROWN\n" );
33       RemoveItemsBetween( list1, "BLACK", "BROWN" );
34
35       PrintList( list1 ); // print list1 elements
36       PrintReversedList( list1 ); // print list in reverse order
37    } // end method Main
38
39    // output list contents
40    private static void PrintList< E >( LinkedList< E > list )
41    {
42       Console.WriteLine( "Linked list: " );
43
44       foreach ( E value in list )
45          Console.Write( "{0} ", value );
46
47       Console.WriteLine();
48    } // end method PrintList
```

Use overloaded constructor to create a new `LinkedList` initialized with the contents of array `color2`

(2 of 5)

The generic method iterates and outputs the values of the `LinkedList`

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
49
50    // concatenate the second list on the end of the first list
51    private static void Concatenate< E >( LinkedList< E > list1,
52       LinkedList< E > list2 )
53    {
54       // concatenate lists by copying element values
55       // in order from the second list to the first list
56       foreach ( E value in list2 )
57          list1.AddLast( value ); // add new node
58    } // end method Concatenate
59
60    // locate string objects and convert to uppercase
61    private static void ToUppercaseStrings( LinkedList< string > list )
62    {
63       // iterate over the list by using the nodes
64       LinkedListNode< string > currentNode = list.First;
65
66       while ( currentNode != null )
67       {
68          string color = currentNode.Value; // get value in node
69          currentNode.Value = color.ToUpper(); // convert to uppercase
70
71          currentNode = currentNode.Next; // get next node
72       } // end while
73    } // end method ToUppercaseStrings
```

Append each value of `list2` to the end of `list1`

`LinkedListTest.cs`

(3 of 5)

Takes in a `LinkedList` of type `string`

Property to obtain the first `LinkedListNode`

Convert each of the `string`s to uppercase

Traverse to the next `LinkedListNode`

© 2006 Pearson Education, Inc. All rights reserved.

Outline

Obtain the “boundaries” nodes of the range

LinkedListTest.cs

(4 of 5)

```
74
75    // delete list items between two given items
76    private static void RemoveItemsBetween< E >( LinkedList< E > list,
77       E startItem, E endItem )
78    {
79       // get the nodes corresponding to the start and end item
80       LinkedListNode< E > currentNode = list.Find( startItem );
81       LinkedListNode< E > endNode = list.Find( endItem );
82
83       // remove items after the start item
84       // until we find the last item or the end of the linked list
85       while ( ( currentNode.Next != null ) &&
86          ( currentNode.Next != endNode ) )
87       {
88          list.Remove( currentNode.Next ); // remove next node
89       } // end while
90    } // end method RemoveItemsBetween
91
92    // print reversed list
93    private static void PrintReversedList< E >( LinkedList< E > list )
94    {
95       Console.WriteLine( "Reversed List:" );
```

Remove one element node at a time and fix the references of the surrounding nodes

© 2006 Pearson Education, Inc. All rights reserved.

```
96
97        // iterate over the list by using the nodes
98        LinkedListNode< E > currentNode = list.Last;
99
100       while ( currentNode != null )
101       {
102          Console.Write( "{0} ", currentNode.Value );
103          currentNode = currentNode.Previous; // get previous node
104       } // end while
105
106       Console.WriteLine();
107    } // end method PrintReversedList
108} // end class LinkedListTest
```

Property to obtain the last `LinkedListNode`

Traverse to the previous `LinkedListNode`

LinkedListTest.cs

(5 of 5)

```
Linked list:
black yellow green blue violet silver gold white brown blue gray

Converting strings in list1 to uppercase

Linked list:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN BLUE GRAY

Deleting strings between BLACK and BROWN

Linked list:
BLACK BROWN BLUE GRAY
Reversed List:
GRAY BLUE BROWN BLACK
```

© 2006 Pearson Education, Inc. All rights reserved.

27.6 Synchronized Collections

- **Synchronization with Collections**
  - **Most non-generic collections are unsynchronized**
    - **Concurrent access to a collection by multiple threads may cause errors**
  - **Synchronization wrappers**
    - **Prevent potential threading problems**
    - **Used for many of the collections that might be accessed by multiple threads**
    - **Wrapper object receives method calls, adds thread synchronization, and passes the calls to the wrapped collection object**
  - **Most of the non-generic collection classes provide `static` method `Synchronized`**
    - **Returns a synchronized wrapping object for the specified object**

```
ArrayList notSafeList = new ArrayList();
ArrayList threadSafeList = ArrayList.Synchronized( notSafeList );
```

  - **The collections in the .NET Framework do not all provide wrappers for safe performance under multiple threads**
  - **Using an enumerator is not thread-safe**
    - **Other threads may change the collection**
    - `foreach` **statement is not thread-safe either**
    - **Use the `lock` keyword to prevent other threads from using the collection**
    - **Use a `try` statement to catch the `InvalidOperationException`**

© 2006 Pearson Education, Inc. All rights reserved.