Enumerators and Enumerables

The IEnumerator Interface

The IEnumerable Interface

The Generic Enumeration Interfaces

Iterators

Common Iterator Patterns

Producing Multiple Enumerables

Iterators As Properties

Behind the Scenes with Iterators
Enumerators and Enumerable Types

In Chapter 12, you saw that you can use a foreach statement to cycle through the elements of an array. In this chapter, you’ll take a closer look at arrays and see why they can be processed by foreach statements. You’ll also look at how you can add this capability to your own user-defined classes, using iterators.

Using the foreach Statement

When you use a foreach statement with an array, the statement presents you with each element in the array, one by one, allowing you to read its value. For example, the following code declares an array with four elements, and then uses a foreach loop to print out the values of the items:

```csharp
int[] arr1 = { 10, 11, 12, 13 }; // Define the array.
foreach (int item in arr1) // Enumerate the elements.
{
    Console.WriteLine("Item value: {0}", item);
}
```

This code produces the following output:

```
Item value: 10
Item value: 11
Item value: 12
Item value: 13
```

Why does this work with arrays? The reason is that an array can produce, upon request, an object called an enumerator. The enumerator is an object that can return the elements of the array, one by one, in order, as they’re requested. The enumerator “knows” the order of the items and keeps track of where it is in the sequence. It then returns the current item when it is requested.

For a type that has an enumerator, there must be a way of retrieving it. The way to retrieve an object’s enumerator is to call the object’s GetEnumerator method. Types that implement a GetEnumerator method are called enumerable types, or just enumerables. Arrays are enumerables.