**Understanding Scope**

Now that we're dividing our code into procedures, it's a good idea to look at the issue of scope because putting code in procedures restricts that code's scope. Now that Visual Basic .NET is emphasizing OOP more than ever before, scope has become even more important because much of the power of classes and objects is all about restricting scope and hiding implementation details to make things simpler.

The *scope* of a programming element in your code is all the code that can access it. In other words, an element's scope is its *accessibility* in your code. In Visual Basic .NET, where you declare an element determines its scope, and an element can have one of the following levels of scope:

* **Block scope**—The item is available only within the code block in which it is declared.
* **Procedure scope**—The item is available only within the procedure in which it is declared.
* **Module scope**—The item is available to all code within the module, class, or structure in which it is declared.
* **Namespace scope**—The item is available to all code in the namespace.

Let's look at these various levels of scope.

**Block Level**

A code block is the body of a compound statement. A compound statement is one that can hold other statements, such as an If statement. Here's an If statement in which a variable, strText, is declared. Note that strText is *inaccessible* outside the If statement, so code that tries to display its value won't work:

Module Module1

Sub Main()

Console.WriteLine

Console.WriteLine("Enter a letter...")

Dim strInput = Console.ReadLine()

If strInput = "q" Then

End

Else

Dim strText As String = "Please type q to quit."

Console.WriteLine(strText)

End If

Console.WriteLine(strText) 'Will not work!

Console.WriteLine("Press Enter to continue...")

Console.ReadLine()

End Sub

End Module

**Procedure Level**

An element declared in a procedure is not available outside that procedure, which means that only the code in the procedure that contains the declaration can access it. Elements at this level are called*local* elements, and you declare them with the Dim or Static statements. In the following example, the variable strText declared in the ShowMessage Sub procedure cannot be accessed in the MainSub procedure:

Module Module1

Sub Main()

ShowMessage()

Console.WriteLine(strText) 'Will not work!

Console.WriteLine("Press Enter to continue...")

Console.ReadLine()

End Sub

Sub ShowMessage()

Dim strText = "Hi there!"

Console.WriteLine(strText)

End Sub

End Module

**Module Level**

Visual Basic .NET uses the term *module level* to apply to three programming elements: modules, classes, and structures. (We'll see classes later today and structures in Day 9.) You declare elements at this level by placing the declaration outside any procedure or block in the module, class, or structure.

Unlike in blocks or procedures (where you can use only Dim or Static), at the module level you can also use these keywords to restrict or enlarge scope. (Don't feel you have to memorize these definitions at this stage; we'll see more on these terms throughout the book.)

* Public—The Public statement declares elements to be accessible anywhere. This includes inside the same project, from other projects that reference the current project, assemblies built from the project, and so on.
* Protected—The Protected statement declares elements to be accessible only from within the same class or from a class derived from this class. You can use Protected only at class level and only when declaring a member of a class.
* Friend—The Friend statement declares elements to be accessible from within the same project, but not from outside the project.
* Protected Friend—The Protected statement with the Friend keyword declares elements to be accessible either from derived classes or from within the same project, or both. You can use Protected Friend only at class level.
* Private—The Private statement declares elements to be accessible only from within the same module, class, or structure.

Let's look at module-level scope with some examples. For example, you can create a new code module, Module2, like this:

Module Module1

Sub Main()

End Sub

End Module

Module Module2

End Module

**TIP**

Although this example declares two modules in the same file (Module1.vb), you can also add a module in a new file to a Visual Basic project by selecting the Project, Add Module menu item (which will create Module2.vb, Module3.vb, and so on).

And if you declare a new Sub procedure, ShowMessage, in the new module, you can access it from the first module:

Module Module1

Sub Main()

ShowMessage()

Console.WriteLine("Press Enter to continue...")

Console.ReadLine()

End Sub

End Module

Module Module2

Sub ShowMessage()

Console.WriteLine("Hello there!")

End Sub

End Module

However, if you declare the Sub procedure Private to the new module, you cannot access it in the first module:

Module Module1

Sub Main()

ShowMessage() 'Will not work!

Console.WriteLine("Press Enter to continue...")

Console.ReadLine()

End Sub

End Module

Module Module2

Private Sub ShowMessage()

Console.WriteLine("Hello there!")

End Sub

End Module

In module scope, you can also make variables—not just procedures—public or private; this example declares strText in the second module using a Dim statement:

Module Module1

Sub Main()

Console.WriteLine(strText) 'Will not work!

Console.WriteLine("Press Enter to continue...")

Console.ReadLine()

End Sub

End Module

Module Module2

Dim strText = "Hello there!"

End Module

By default, module-level variables are declared Private when you use Dim, so strText cannot be accessed outside its module. However, if you declare this new variable Public, it can be accessed in the first module with no problem:

Module Module1

Sub Main()

Console.WriteLine(strText)

Console.WriteLine("Press Enter to continue...")

Console.ReadLine()

End Sub

End Module

Module Module2

Public strText = "Hello there!"

End Module

**Namespace Scope**

You can also declare elements at namespace level in Visual Basic. A *namespace* is an OOP feature used to keep elements with the same name from conflicting with each other in larger programs. (If you don't use a Namespace statement in your code, all your code is in the same namespace.) Declaring a module-level element Friend or Public makes it available to all procedures throughout the namespace.

We now have the background we'll need on procedures and scope, two very important programming concepts. Next, let's tackle handling the runtime errors that may crop up because the Visual Basic language puts special emphasis on this topic.