**Assemblies and the internal Access Modifier**

**Assemblies:**

* An integral part of C# programming is the assembly.
* An assembly is a file (or files) that contains all deployment and version information for a program.
* Assemblies are fundamental to the .NET environment. They provide mechanisms that support safe component interaction, interlanguage operability, and versioning.
* An assembly also defines a scope.
* An assembly is composed of four sections.
* The first is the assembly manifest. The manifest contains information about the assembly, itself. This data includes such things as the name of the assembly, its version number, type mapping information, and cultural settings.
* The second section is type metadata, which is information about the data types used by the program. Among other benefits, type metadata aids in cross-language interoperability.
* The third part of an assembly is the program code, which is stored in Microsoft Intermediate Language (MSIL) format.
* The fourth constituent of an assembly is the resources used by the program.
* Fortunately, when using C#, assemblies are produced automatically, with little or no extra effort on your part. The reason for this is that the exe file created when you compile a C# program is actually an assembly that contains your program’s executable code as well as other types of information.
* Thus, when you compile a C# program, an assembly is automatically produced.
* There are many other features and topics that relate to assemblies, but a discussion of these is outside the scope of this book. (Assemblies are an integral part of .NET development, but are not technically a feature of the C# language.)
* However, there is one part of C# that relates directly to the assembly: the internal access modifier, which is examined next.

**The internal Access Modifier:**

* In addition to the access modifiers public, private, and protected, which you have been using throughout this book, C# also defines internal.
* The internal modifier declares that a member is known throughout all files in an assembly, but unknown outside that assembly.
* Thus, in simplified terms, a member marked as internal is known throughout a program, but not elsewhere.
* The internal access modifier is particularly useful when creating software components.
* The internal modifier can be applied to classes and members of classes and to structures and members of structures.
* The internal modifier can also be applied to interface and enumeration declarations.
* You can use protected in conjunction with internal to produce the protected internal access modifier pair.
* The protected internal access level can be given only to class members.
* A member declared with protected internal access is accessible within its own assembly or to derived types.

Here is an example that uses internal:

// Use internal.

**using System;**

**class InternalTest**

{

Internal int x;

}

**class InternalDemo**

{

**public static void Main(String[] args)**

{

**InternalTest ob = new InternalTest();**

**ob.x** = 10; // can access -- in same file

Console.WriteLine("Here is ob.x: " + ob.x);

}

}

* Inside InternalTest, the field x is declared internal.
* This means that it is accessible within the program, as its use in InternalDemo shows, but unavailable outside the program.