**Value and Reference Types**

C# supports two kinds of variable types:

* Value types

These are the built-in primitive data types, such as char, int, and float, as well as user-defined types declared with struct.

* Reference types

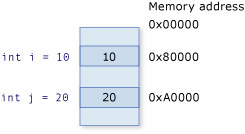
Classes and other complex data types that are constructed from the primitive types. Variables of such types do not contain an instance of the type, but merely a reference to an instance.

If you create two value-type variables, i and j, as follows, then i and j are completely independent of each other:

int i = 10;

int j = 20;

They are given separate memory locations:



If you change the value of one of these variables, the other will naturally not be affected. For instance, if you have an expression such as the following, then there is still no connection between the variables:

int k = i;

That is, if you change the value of i, k will remain at the value that i had at the time of the assignment.

i = 30;

System.Console.WriteLine(i.ToString()); // 30

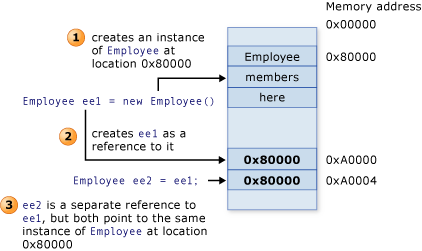
System.Console.WriteLine(k.ToString()); // 10

Reference types, however, act differently. For instance, you could declare two variables as follows:

Employee ee1 = new Employee();

Employee ee2 = ee1;

Now, because classes are reference types in C#, ee1 is known as a reference to Employee. The first of the previous two lines creates an instance of Employee in memory, and sets ee1 to reference it. Thus, when you set ee2 to equal ee1, it contains a duplicate of the reference to the class in memory. If you now change properties on ee2, properties on ee1 reflect these changes, because both point to the same object in memory, as shown in the following:



http://i.msdn.microsoft.com/Global/Images/clear.gif Boxing and Unboxing

The process of converting a value type to a reference type is called boxing. The inverse process, converting a reference type to a value type, is called unboxing. This is illustrated in the following code:

int i = 123; // a value type

object o = i; // boxing

int j = (int)o; // unboxing