Passing Parameters by Value and by Reference

Consider the following method:

```
static int doubleValue( int n )
{
   return (n * 2);
}
```

Look closely at the first line of the method:

```
static int doubleValue( int n )
```

This line declares that the method *doubleValue* takes an integer parameter, which we will refer to as *n* inside of the method, and that it will return an integer value. The body of the method is written between the curly braces. To call this method, we could write something like

```
int m = doubleValue( 5 );
```

This statement would pass the value of 5 as the argument to the method, and the return value of the method, 10, will be stored in the variable m when the method is done. Values are passed to methods on what is called the **run-time stack**. If a function does not return a value, we denote this by using the keyword **void**.

The following code illustrates they way that we write and call methods in a program.

In this example, the parameter *n* is passed **by value**. Consider the first time that the function is called. At that moment, the variable *i* contains the value of 0. The compiler makes a copy of this value and places it on the run-time stack. Inside the function, the value is taken from the stack and stored in the local variable *n*. When we double *n*, it only affects the variable *n* that is defined local to the function. The original variable *i* is not changed.

The following example illustrates a complete program that includes this method.

```
using System;

class Program
{
   const int SIZE = 5;

   static void Main( )
```

```
      {
         for (int i = 0; i < SIZE; i++)
         {
            Console.WriteLine( DoubleValue( i ) );
         }
      }

      // The DoubleValue method
      // Purpose: doubles the value passed as a
parameter
      // Parameters: an integer value
      // Returns: an integer valuebr>       //
Preconditions: none
      // Postconditions: The return value is twice the
parameter
      static int DoubleValue( int n )
      {
         return (n * 2);
      }
   }
```

Data can also be passed to a method **by reference**. When an argument is passed **by reference**, the compiler puts a handle to the argument on the run-time stack, not a copy of its value. Now, when the method uses the argument, it uses the actual variable whose handle was passed on the stack, not a copy of the variable. We denote a pass by reference by using the keyword **ref** before the type name in the method's signature. For example:

```
   static int doubleValue( ref int n );
```

We must also use the ref keyword when calling a method that requires a reference parameter. For example

```
   int x = DoubleValue( ref myValue );
```

Because we have given the method a handle to a variable declared somewhere outside of the method, the method is free to change the value of that variable. Usually, this is not a desirable thing to do. Changing the value of a variable declared outside of the method is called a **side effect**. In general, we do not like to have side effects in our code. To be more precise, a method should only change the values of variables that are declared inside of the method. However, there are exceptions to this rule.