**Use Outer Variables with Anonymous Methods**

A local variable or parameter whose scope includes an anonymous method is called an

*outer variable.* An anonymous method has access to and can use these outer variables. When

an outer variable is used by an anonymous method, that variable is said to be *captured.* A

captured variable will stay in existence at least until the delegate that captured it is subject

to garbage collection. Thus, even though a local variable will normally cease to exist when

its block is exited, if that local variable is being used by an anonymous method, then

that variable will stay in existence at least until the delegate referring to that method is

destroyed.

The capturing of a local variable can lead to unexpected results. For example, consider

this version of the counting program. As in the previous version, the summation of the

count is computed. However, in this version, a **CountIt** object is constructed and returned

by a static method called **Counter( )**. This object uses the variable **sum**, which is declared in

the enclosing scope provided by **Counter( )**, rather than in the anonymous method, itself.

Thus, **sum** is captured by the anonymous method. Inside **Main( )**, **Counter( )** is called to

obtain a **CountIt** object. Thus, **sum** will not be destroyed until the program finishes

As you can see, the count still proceeds normally. However, notice the summation value for

5. It shows 21 instead of 15! The reason for this is that **sum** is captured by **ctObj** when it is

created by the **Counter( )** method. This means it remains in existence until **count** is subject

to garbage collection at the end of the program. Thus, its value is not destroyed when

**Counter( )** returns or with each call to the anonymous method when **count** is called in **Main( )**.

Although captured variables can result in rather counterintuitive situations, such as the

one just shown, it makes sense if you think about it a bit. The key point is that when an

anonymous method captures a variable, that variable cannot go out of existence until the

delegate that captures it is no longer being used. If this were not the case, then the captured

variable could be undefined when it is needed by the delegate.