Classes, Objects, and References

After a class has been defined, you may allocate any number of objects using the C# new keyword.

Understand, however, that the new keyword returns a ***reference* to the object on the heap**, not the actual

object itself.

If you declare the reference variable as a local variable in a method scope, it is stored on the stack for further use in your application. When you want to invoke members on the object, apply the C# dot operator to the stored reference



The Basics of Object Lifetime

When you are building your C# applications, you are correct to assume that the .NET runtime environment (a.k.a. the CLR) will take care of the managed heap without your direct intervention. In fact, the golden rule of .NET memory management is simple:

 **Rule** Allocate a class instance onto the managed heap using the new keyword and forget about it.

Once instantiated, the **garbage collector will destroy an object** when it is no longer needed. The next obvious question, of course, is, “How does the garbage collector determine when an object is no longer needed?” The short (i.e., incomplete) answer is that the garbage collector removes an object from the heap **only if it is *unreachable* by any part of your code base.**

the managed heap maintains a pointer (commonly referred to as the *next*

*object pointer* or *new object pointer*) that identifies exactly where the next object will be located. That

said, the newobj instruction tells the CLR to perform the following core operations:

• Calculate the total amount of memory required for the object to be allocated

(including the memory required by the data members and the base classes).

• Examine the managed heap to ensure that there is indeed enough room to host

the object to be allocated. If there is, the specified constructor is called and the

caller is ultimately returned a reference to the new object in memory, whose

address just happens to be identical to the last position of the next object pointer.

• Finally, before returning the reference to the caller, advance the next object

pointer to point to the next available slot on the managed heap.



As your application is busy allocating objects, the space on the managed heap may eventually become full. When processing the newobj instruction, if the CLR determines that the managed heap does not have sufficient memory to allocate the requested type, it will perform a garbage collection in an attempt to free up memory. Thus, the next rule of garbage collection is also quite simple:

 **Rule** If the managed heap does not have sufficient memory to allocate a requested object, a garbage collection will occur.

During a garbage collection process, the runtime will investigate objects on the managed heap to

determine whether they are still reachable (i.e., rooted) by the application. To do so, the CLR will build

an *object graph*, which represents each reachable object on the heap.

Assume the managed heap contains a set of objects named A, B, C, D, E, F, and G. During a garbage collection, these objects (as well as any internal object references they may contain) are examined for active roots. After the graph has been constructed, unreachable objects (which we will assume are objects C and F) are marked as garbage. Figure 13-3 diagrams a possible object graph for the scenario just described (you can read the directional arrows using the phrase *depends on* or *requires*, for example,

E depends on G and B, A depends on nothing, and so on).



