## Long-Lived Objects

Only a full garbage collection will take care of an object that has made it to Gen 2, and thus these objects can possibly consume memory long after they are dead. It is therefore best to make sure that the size of each of these objects is kept to a minimum.

Inefficient code can make objects live longer than they should, reaching Gen 2 and wasting memory unnecessarily. Avoid this by only allocating objects you need, and letting go of references when they're no longer necessary: references on the stack cause rooted objects for the GC. I often review code that has been rewritten several times and has had no attention paid to cleanup. It does not take much time for well-structured code to degenerate into a mess.

Keep your methods small, create objects as close as possible to when you're going to use them, and always remember to call `Dispose()` on your disposable objects. If an object is created before a slow method call and won't be needed after it returns, release the object before calling the method. If the GC runs while this long-running process is running, the object will be promoted to a later generation. Objects take longer to collect.