9. What are generations?

Brief summary: The Garbage Collector divides objects into three generations - 0, 1, and 2 - depending on their longevity. Short-lived objects belong to generation 0, and if they survive their first collection, they are moved to generation 1, and after that - to generation 2. The Garbage Collector collects objects from generation 0 most often, and from generation 2 least often. This feature is introduced in order to improve Garbage Collector's performance. Objects that survived a couple of cycles of the GC's work tend to be long-lived and they don't need to be checked upon so often. This way, the Garbage Collector has less work to do, so it can do it faster.

Garbage Collector has a lot of work. It needs to identify objects to be removed, which we learned about in the previous lecture. Then, it must actually remove them. Finally, it must defragment the memory of the application.

As we know the Garbage Collector marks objects as reachable or unreachable and removes the latter from the memory. Imagine there is some object - let's call it object A. During the first execution of the Garbage Collector, this object is marked as reachable, and so are all objects reachable from object A. After some time the Garbage Collector gets to work again, and again, it marks object A and its friends as reachable. Then again, some time passes, and Garbage Collector gets triggered again… and again it marks object A as reachable.

If I were the Garbage Collector, I would probably get a bit frustrated. This object is obviously long-lived, and I don't want to check if it's still needed every time I get to work! I have other things to do!

Well, that's exactly the optimization the Garbage Collectors creators come up with - to not make the Garbage Collector check each object every time. If some object "survived" a couple of cycles of the Garbage Collector's work, it's most likely long-lived, and we should check it only every once in a while.

This optimization introduced the concept of **generations** of objects. Once an object is first created, it is assigned to generation 0. If it survives its first collection, it advances to generation 1. If it survives the second, it advances to generation 2.

The Garbage Collector checks objects from Generation 0 most frequently. Less frequently it checks objects from generation 1, and least frequently - from generation 2. It makes sense. If the object survived two collections, it's most likely

long-lived, and there is a big chance it will survive collections number 10, 20, or 100. We don't need to check on it that often.

For example, think of some logger objects. Often there is one logger object created at the start of the program execution, and it is passed around to any class that needs it. Its life cycle is basically as long as the time the application runs.

As the opposite, we often create objects that last only for a very short time, like anonymous objects created to temporarily carry some data between LINQ queries. In this case, it is reasonable that the Garbage Collector will quickly remove them, so they don't occupy memory for too long.

In a well-tuned application, most objects die in generation 0.

Please note that during a collection of a generation, all previous generations are also collected. So when generation 0 is collected, no other one is, but when generation 2 is collected, generations 0 and 1 are too (that's why collecting objects from generation 2 is sometimes called "a full garbage collection").

One more thing that needs to be mentioned is the LOH - the Large Objects Heap. When a very large object (larger than 85 000 bytes) is initially created, it is stored in a special area of memory called the **Large Objects Heap**, and it is assigned to generation 2 right away. It gets this special treatment because it rarely happens that very large objects are short-lived. Also, the objects in the Large Objects Heap have one more special feature - they are **pinned**. It means, they will not be moved in memory during the defragmentation step of the Garbage Collector's work, which happens after removing unreferenced objects from memory. This is because the larger the object is, the more expensive is the operation of moving it (and the harder it is to find the chunk of memory large enough to fit it). It's better to "pin" such large objects, and move smaller objects around them. (FYI, starting with .NET 4.5.1 we can change this default and "unpin" the pinned objects living in the LOH).

Let's summarize. The Garbage Collector divides objects into three generations - 0, 1, and 2 - depending on their longevity. Short-lived objects belong to generation 0, and if they survive their first collection, they are moved to generation 1, and after that - to generation 2. The Garbage Collector collects objects from generation 0 most often, and from generation 2 least often. This feature is introduced in order to improve Garbage Collector's performance. Objects that survived a couple of cycles of the GC's work tend to be long-lived and they don't need to be checked upon so often. This way, the Garbage Collector has less work to do, so it can do it faster.

Bonus questions:

- "**What is the Large Objects Heap?**"
  It's a special area of the heap reserved for objects larger than 85 000 bytes. Such objects logically belong to generation 2 from the very beginning of their existence and are pinned.

- "**What does it mean that the object is pinned?**"
  It means it will not be moved during the memory defragmentation that the Garbage Collector is executing. It is an optimization, as large objects are expensive to move, and it's hard to find a chunk of memory large enough for them.