

Re: C# confusion

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2005-12/msg02402.html>

- *From:* "Bruce Wood" <brucewood@xxxxxxxxxxx>
 - *Date:* 12 Dec 2005 11:22:52 -0800
-

Jesika wrote:

- > I'm coming from a 3 year C background with some basic knowlegde on the Win32
- > API. I just started learning C# a day ago and I have a couple questions that
- > I need cleared up that just have me very confused.
- >
- > [quesitons]
- >
- > 1.) Whenever you declare a class – MyClass – is the class alive in memory
- > for the whole duration of the program? If so, then, since it was not created
- > via the "new" keyword (like an instance is), is it on the heap or stack?

You have to distinguish here between three things. First, you are essentially correct (with caveats following): if you never say "new MyClass" then you will never create an instance of MyClass on the heap, so any instance members you declare in MyClass will never exist. That said, you could declare some things in MyClass as "static", meaning that there is only one copy for MyClass as a whole, shared amongst all instances. I believe that static members are stored in a separate area of memory: not the stack and not the heap, since they are brought into existence the first time anything in MyClass is referenced by your program, and are never garbage collected. Ever. Third, remember that classes consist of data and code, so the `_code_` associated with MyClass will in fact take up some memory, but again it is static in nature so it lives neither on the stack nor on the heap.

The memory allocated for instance members of classes is always stored in the heap. Local variables and parameters to methods are stored on the stack.

- > 2.) If you create instances of MyClass, you're really just creating copies
- > of MyClass, like you would create instances of a struct (non-pointer structs)
- > in C/C++, correct?

You're creating in the heap copies of the instance members you declared for MyClass, yes. Note that the code for MyClass any any static members are not copies over and over again on the heap. They reside in static memory.

Re: C# confusion

- > 3.) When MyClass is not declared inside another class, or namespace for that matter, why is it that you cannot make MyClass private? Is it because it would make the class totally useless since you would not be able to create any instances of it or use it via the MyClass type?

Exactly. The compiler won't let you declare something that you can't use.

- > 4.) Still pertaining to the above; if in the programs duration, you never invoke MyClass or create any instances of it, is it still created in memory or is it **only** created when you create an instance of it or try to access it via the MyClass type? if it's the latter, would it make any difference if any member in the class or the class itself was made static?

If you never reference anything about MyClass, I believe that the code still occupies memory space. I don't know whether the IL (Intermediate Language) is JITted (just-in-time compiled) or not. Anyone? At what granularity does the JITter work? Does it always compile entire assemblies? Or just entire classes? Or method-by-method as needed?

However, you are essentially correct: if you never make any reference to MyClass, even its static members will never be initialized. If you never say "new MyClass", then MyClass will never occupy any heap space.

- > 4.) static keyword; This one has been hitting me like a hammer on the head. Ok, from what I can derive from my reading so far is that the concept of the static keyword on members is like using global variables or passing pointers around to functions in C/C++, Am I correct? Is that its main and/or sole purpose?

Semantically, "static" provides a way to associate a variable with the class as a whole, rather than a separate copy of the variable for each instance. A classic application of this is the Singleton design pattern. C also has a "static" keyword which, although it doesn't translate directly, had the same sort of memory behaviour: static variables persisted across method calls, and so were stored neither on the heap nor on the stack.

Static is sort of like having a global variable, but it is more controlled and safer. For one thing, you can make it local to your class, strictly for internal use, so it's "global" to the class, but not global to your program. It's also necessary in certain circumstances, for example creating a cache to speed up certain methods in your class. You want the cache to be shared amongst all instances of the class rather than having each instance have its own cache.

However, "static" is not like passing pointers around in C / C++. Passing pointers as arguments means putting the pointer on the stack. Static members live in static memory, not stack memory, so they're very different from pointer arguments.

-
- Prev by Date: [*Re: C# confusion*](#)
 - Next by Date: [*QueueUserWorkItem: Design guidance*](#)
 - Previous by thread: [*Re: C# confusion*](#)
 - Next by thread: [*QueueUserWorkItem: Design guidance*](#)
 - Index(es):
 - ◆ [*Date*](#)
 - ◆ [*Thread*](#)