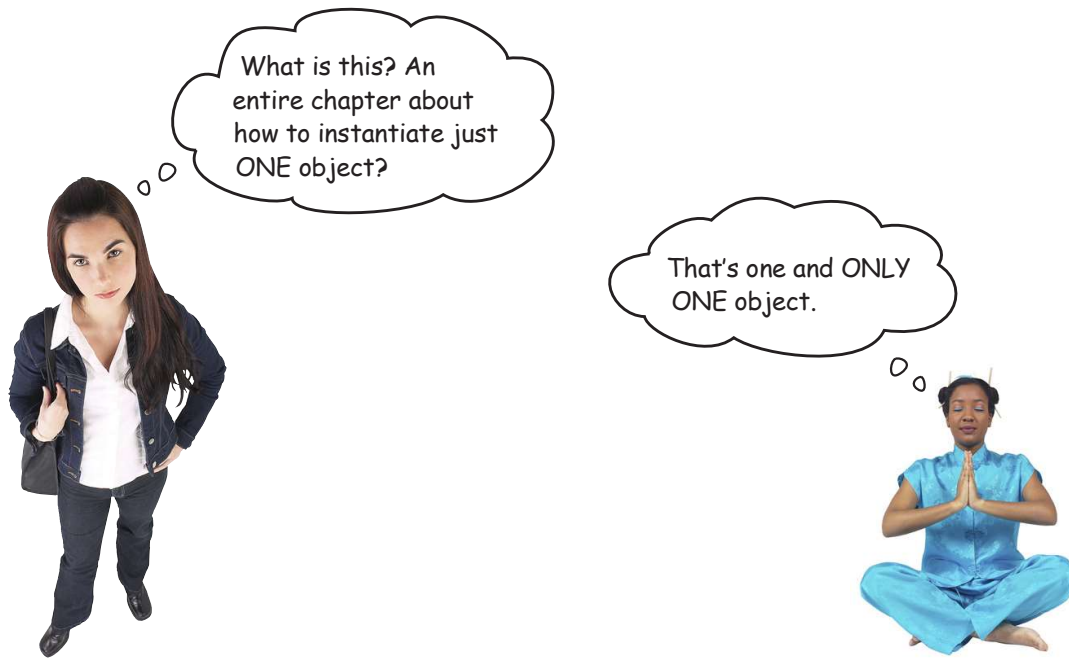


one and only one



Developer: What use is that?

Guru: There are many objects we only need one of: thread pools, caches, dialog boxes, objects that handle preferences and registry settings, objects used for logging, and objects that act as device drivers to devices like printers and graphics cards. In fact, for many of these types of objects, if we were to instantiate more than one we'd run into all sorts of problems like incorrect program behavior, overuse of resources, or inconsistent results.

Developer: Okay, so maybe there are classes that should only be instantiated once, but do I need a whole chapter for this? Can't I just do this by convention or by global variables? You know, like in Java, I could do it with a static variable.

Guru: In many ways, the Singleton Pattern is a convention for ensuring one and only one object is instantiated for a given class. If you've got a better one, the world would like to hear about it; but remember, like all patterns, the Singleton Pattern is a time-tested method for ensuring only one object gets created. The Singleton Pattern also gives us a global point of access, just like a global variable, but without the downsides.

Developer: What downsides?

Guru: Well, here's one example: if you assign an object to a global variable, then that object might be created when your application begins. Right? What if this object is resource intensive and your application never ends up using it? As you will see, with the Singleton Pattern, we can create our objects only when they are needed.

Developer: This still doesn't seem like it should be so difficult.

Guru: If you've got a good handle on static class variables and methods as well as access modifiers, it's not. But, in either case, it is interesting to see how a Singleton works, and, as simple as it sounds, Singleton code is hard to get right. Just ask yourself: how do I prevent more than one object from being instantiated? It's not so obvious, is it?