GC.SuppressFinalize(this);

you're telling the system that whatever work would have been done in the finalizer has already been done, so the finalizer doesn't need to be called. From the .NET docs:

Objects that implement the IDisposable interface can call this method from the IDisposable.Dispose method to prevent the garbage collector from calling Object.Finalize on an object that does not require it.

**Object**

An actual type, like any other type, with one special rule: if a type doesn't inherit, it inherits from *object*. From this, it follows that *all* types inherit from *object*, directly or indirectly.

Emphasis: **object is a type**. An object can be of type *object*, and the type has its methods, like ToString(). Since everything inherits from *object*, everything can be upcast into *object*. When you assign an object to an *object* reference, you are doing upcasting just like when you assign an *Elephant* type object to an *Animal* reference where *Elephant* inherits from *Animal*.

* *obj* is treated as being of type *object* at compile time, and will be of type *object* at runtime (which is logical, since it is an actual type - *obj* is declared as *object* so can only be of that type)
* *obj.DoSomething()* will cause a compile-time error, as *object* does not have this method, regardless of whether SomeType has it.

**Var**

This is not an actual type, it is merely shorthand for "compiler, figure out the type for me based on the right side of the assignment".

* *obj* is treated as being of type *SomeType* at compile time, and will be of type *SomeType* at runtime, just as if you had written "SomeType" instead of "var".
* if *SomeType* has a method *DoSomething()*, this code will work
* if *SomeType* doesn't have the method, the code will cause a compile-time error

**Dynamic**

This is a type that tells the compiler to disable compile-time type checking on the variable. An object is treated as having the type *dynamic* at compile-time and run-time.

* obj is of type *dynamic* at compile and run time
* if *SomeType* has a method *DoSomething()*, this code will work
* if *SomeType* doesn't have the method, the code will compile, but throw an exception at run-time
* note that *dynamic* can cause exceptions very easily if used carelessly:
* public void f(dynamic x)
* {
* x.DoSomething();
* }

This will throw an exception if x is of a type that doesn't have the DoSomething method, but it will still be possible to call it and pass any object as the parameter without a compile-time error, causing an error that only shows itself at run-time, and possibly only in specific circumstances - a potential bug. So if you use dynamic in any kind of public interface of a class, you should always manually type-check at runtime using reflection, carefully deal with exceptions, or not do it in the first place.

Note: the object being referred to never changes its type, of course. While obj may be object, the xthat it refers to is still SomeType.

|  |
| --- |
|  |
|  |  |
|  |  |