

Re: Abstract class variables question

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-11/msg02076.html>

- *From:* Peter Duniho <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 16 Nov 2007 00:33:03 -0800
-

On 2007-11-15 21:46:48 -0800, "tshad" <tfs@xxxxxxxxxxxxxxxxxxx> said:

I always looked at a property as a conduit to some value but that it didn't do anything until you called it (get). So it was confusing that the value seemed to be "something" before being called by the debugger. But in actuality, I guess you could say the debugger was doing a "get" to get the value.

Not only could I say that, I would. :)

That's how properties work. You have to call the getter method to get the value. Since calling a getter is an instantaneous event, rather than something that's stored somewhere, a property's value is only known at the time you call the getter.

Some properties never change in value, such as the one we're talking about here. Others are more conventional, and change in value only when you call the setter. And still others are changing in value all the time (at this extreme, see for example `DateTime.Now`).

The one thing that all properties have in common is that whatever one might consider the "value" of a property, the only way to know for sure what the value is at any given moment is to call the getter for the property. And the instance you call that getter, there are no fundamental guarantees about what might be returned the next time you call the getter (though many classes of course provide a guarantee above and beyond what the language offers).

To me, it's sort of like a little Heisenberg thing going on. You have to measure the state of the property to know its value. Granted, for most properties measuring the state doesn't itself necessarily change the state, but it could and more importantly you still can't say anything about the future state of the property even having obtained the state at some instantaneous point in time. :)

I'm not really sure what you mean. An object reference (given the .NET terminology, I prefer to use the word "reference" over "pointer" in this situation) points (or rather, refers) to the data structure. It wouldn't "point" to a specific property.

Re: Abstract class variables question

What I was saying was that when you do the line "`_objCurrent = value;`", value can be anything (bool, int, string, etc). To the class it is just an object (not a bool, int or string). The statement doesn't care, which is why you need to do the `ValidateType` before. The value really gets put on the heap and `_objCurrent` is just a pointer to that location. Right?

Basically, yes.

I only sort of understand boxing.

If I understand it correctly, `_objCurrent` is just a pointer on the Stack that points to something on the Heap.

Not really. `_objCurrent` is a field inside a class. All classes are reference types, which means that all classes are allocated from the heap. So `_objCurrent` is a memory location in the heap, that references something else that's stored in the heap.

When I assign true to an object, it does

- 1) a "memalloc" (from the C++ days) and grabs a portion of memory
- 2) puts the value (true in this case, "This is a test" in the case of a string) there
- 3) then puts the pointer (reference) in `_objCurrent`.

That much is basically correct, not counting the hand-waving. :) There is more to boxing though than just allocating a large enough area to hold the value itself. It's oversimplifying things to say that just the value is stored (that is, a boxed int for example is going to take more space than just the 32 bits for the int).

Well, all that `_objCurrent` "knows" is that it has a reference to something that is an Object. It doesn't even know that the object has a value type boxed inside it, never mind what the actual value of that value type is.

What is confusing is that Data is type "object", as you said. How does it know how much memory to allocate?

Because a reference is always just 32 bits. The act of boxing a value type needs to know how much space to allocate, to contain the value type along with the class that's containing it. But once the value type has been boxed (and this occurs `_before_` the assignment to the Data property), it's just a regular reference and always only takes as much space to store as any reference: 32 bits.

(Ignoring the possibility of a 64-bit .NET for the moment :))

Re: Abstract class variables question

I assume that the Property knows how much data is being passed. Obviously, a string would/could take many bytes. We do the `_ValidateType` to check the type, but then we just move the value (which could be any type) to the heap.

Well, a string is even easier, since the `String` class is already a reference type. No boxing has to happen; the reference that is already a reference to the string instance is simply copied to the `_objCurrent` field that is the backing store for the `Data` property.

But again, the property does know how much data is being passed, because that data is always 32 bits.

[...]

Actually, I do have a `Nullhandling` class that does just like my last class that we are working on and I plan on using the same techniques we did here to make it work. I use it for moving data to and from my database records and I use these classes to handle the nulls and track changes.

The call to move data from a database to my classes would be something like:

```
NullHandler.GetValueFromDBObject((object)user["UserID"],ref userID);
```

where `userID` is defined as:

```
private IntType userID = new IntType();
```

Well, the first thing I notice is that you're using the `"ref"` keyword inappropriately. The `"userID"` variable is already a reference type. So the parameter is passing a reference. The reference is passed by value, but it's still a reference and so even without the `"ref"` any changes you make to the object in the called method will be reflected when accessing the instance through `"userID"`.

See Jon Skeet's most excellent article on the topic:
<http://www.yoda.arachsys.com/csharp/parameters.html>

Since a lot of the data is the same I was looking at whether to do the same type of thing. I'm not sure if it is worth it or not as it does work as it is but I may do it just for the practice.

Well, at the very least it does seem like you could consolidate all of the `"check for null, convert to type"` code somehow. It sure seems like all of those `GetValueFromDBObject()` overloads could be coalesced into a single method that handles the checking for null.

The way you've written that code, it appears that the caller makes the determination of type before making the call. So one approach would be to add a conversion method into your type classes (similar to the `_TypeRequired` property, it would be abstract and then implemented by each concrete type), instantiate the necessary class before calling `GetValueFromDBObject()`, and then in that method call the conversion method to convert from the database object to the necessary `C#` type for storage.

Re: Abstract class variables question

I don't know enough about the overall design to know that's the best way to handle it, but given the code you've posted so far, it's certainly one way and would at least be better than having all those overloads. :)

Pete

.