

Re: Trying to understand the purpose of interfaces

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-11/msg01550.html>

- *From:* "jm" <needin4mation@xxxxxxxxx>
 - *Date:* 8 Nov 2006 07:08:32 -0800
-

Bruce Wood wrote:

jm wrote:

Do interfaces get used in ASP.NET or is that more of an executable world?

Well, I don't write for ASP.NET, but yes, they would be used there, too.

Rather than talking about ducks, I'd rather point out a classic use of interfaces: your business layer needs to talk to a data layer in order to persist / fetch business information. Typically what one does here is write a bunch of Data classes that mediate between the business layer objects and their representation in a SQL database. No problem so far.

Now say that I want to demo my product on the road, and I don't necessarily have access to a SQL database. I could, of course, use a lightweight SQL database product on my laptop, but another option is to write a whole other data layer that doesn't use a relational database on the back end. Perhaps I want to store some demo data in XML files and read from them instead. How would I do that?

Well, my SQL database persistence classes have their own inheritance hierarchy: CustomerSqlData, say, inherits from SqlDataHandler, which has lots of functionality common to classes that need to talk to SQL databases. If I inherit CustomerXmlData from SqlDataHandler, or from CustomerSqlData, then I get a bunch of code that has to do with SQL databases that I don't want. If I create a whole second inheritance hierarchy—CustomerXmlData inherits from XmlDataHandler—then how do I tell my business layer that CustomerSqlData and CustomerXmlData are really the same thing, just implemented in two different ways?

Interfaces are the answer. I create an interface called ICustomerData, and simply tell my business layer class that it is talking to an

Re: Trying to understand the purpose of interfaces

ICustomerData object. It doesn't need to know which one. The objects don't need to be related by inheritance. In effect, my business layer object is specifying a `_contract_`: "I will deal with any object that implements the following functionality..." rather than specifying a particular `_class_` with which it will work. This frees you to implement the required functionality in any class, anywhere in the inheritance hierarchy, and effectively you can "plug and play" your data layer into your business layer. You can even plug in some data classes that read from XML and some that read from SQL if that turns out to be useful.

Interfaces, then, give you the capability to "chunk off" parts of your application and build plug-and-play parts to implement the various chunks. Each layer of your application knows only that the object it's talking to implement the required contract (interfaces).

This sort of layering and plug-and-play capability applies equally well to ASP.NET and WinForms.

"The objects don't need to be related by inheritance. In effect, my business layer object is specifying a `_contract_`: "I will deal with any object that implements the following functionality..."

Very helpful. Thank you again.

One of the things that was baffling me was constantly thinking about inheritance. All of a sudden interfaces broke that "rule" that was in my mind. If I understand correctly, I can now think of related behaviors and properties that objects have and not strictly "is-a" relationships and inheritance.

But one thing that doesn't make total sense to me is that if I have ClassA and ClassB is derived from ClassA, then I can do things like:

```
ClassB b = new ClassB();  
ClassA a = b; //I thought I could do this anyway
```

because ClassB is a ClassA.

With interfaces, I can now assign objects that implement the same interface to a reference type of that interface. That throws me because the objects are not really related except by behavioral aspects. If ClassA and ClassB are not "kin," but both implement `IMyInterface`, then I can assign them to a reference of type `IMyInterface`. I know it is allowed, but it seems the total opposite of all the other "is-a" class stuff I already had neatly tucked away with the whole inheritance thing. On the other hand, if I go to use these items that implement the same interface, what am I saying other than I want to use the methods they all have in common, which does make sense. Perhaps it is two sides of the coin of inheritance.

Re: Trying to understand the purpose of interfaces

Don't worry if you don't understand what I'm saying. I'm more rambling than anything. It basically says, I think, that it kind of blows up the inheritance model to assign unrelated (by inheritance) items to the same reference type (yet they are sort of related by behavior)..

.