IS JAVA THREATENED?

James W. Cooper

Yesterday, I got a copy of a Sharper Image catalog that featured among the electronic gimcracks a bottle of Scientific Theory diet supplement pills. These pills are billed as a "multi-antioxidant formulation," which I guess means they would be ideal if you had just accidentally swallowed a bottle of Clorox. Never mind the pharmacies, the health food stores and the e-drug companies. Now you can buy these preparations of dubious value right along with your GPS's and weather radios.

Obviously these guys can't distinguish between a theory, an hypothesis and a crackpot idea. A theory is a scientific idea that has been well tested by experiment and observation. Ideas that reach the status of theory (like gravitational theory) would be called facts by most people. An hypothesis is an idea that is put forth for others to test. And a crackpot idea? Well, there are plenty of those in the popular press every day. Even in the computer press.

For example, Microsoft seems to be having a similar positioning problem with its new language C#. They want you to pronounce it "C-sharp." Even though I studied music, my first impression was "C-number" or maybe "C-pound." Since it is apparently supposed to be a "Me Too Not Java" language, we could also say "Pound sand." Is C-sharp better than Java-flat? See C-sharp pound Java flat? Well, that's their, uh, hypothesis. Definitely not a theory.

Actually, Microsoft has announced but not yet shipped two languages that borrow heavily from the successful ideas in Java. The next version of Visual Basic (in Visual Studio 7) finally includes inheritance and a few other Java-like features. Since both will have some impact on the work we do in the next year or so, I looked into them to see how they stacked up.

VB7's Challenge to Java

Visual Basic is the most successful language in the world. Microsoft has shipped over 25 million copies, and claims with some justification that more programmers write programs in Visual Basic than in all other languages (on all platforms) combined. VB is a great language for building Windows user interface-laden programs, It provides a great GUI designer, easy event-driven programming and, since version 3, simple access to databases. It compiles your code into executables that run pretty fast, and it comes with an installation wizard to help users install all the needed DLLs a program needs.

Version 4 of VB introduced class modules, which really began its move to OO programming, VB classes have public and private variables and methods, and you can have many instances of a class. Starting with version 5, VB introduced interfaces, which in a convoluted way allow you to write programs that all are members of the same base class by implementing that class's interface. With that improvement, you could write some pretty good OO code, although I suspect very few actually do so, since there is little literature and few examples of how or why you should write VB code this way. Further, since it is so easy to write VB code that works, there is less impetus to write in a more elegant coding style.

Now, Microsoft has announced that VB7 will finally support inheritance. Interestingly enough, a lot of the original hype on this forthcoming version seems to have dissipated, and there are just a couple of articles on the MS web site now on VB7, and they are caveat-rich documents, indeed.

However, what I can glean from them is that VB7 will add an Inherits keyword that allows any class to inherit the methods of a parent class. Then it can override some or all of those methods, and presumably call the parent methods as well. Here the syntax has not yet been revealed.

Just as interesting is that VB7 will support exceptions. The current error handling (on local error goto) is pretty hard to use well and can lead to errors not being caught where you think they should be if one subroutine calls another and the outer routine discards the error with

```
on local error resume next
```

So this is bound to be an improvement. Of course, it won't really help the millions of lines of legacy VB code, nor that ingrained habits of all the experienced VB programmers, so any change in the area of either inheritance or exception handling is likely to be adopted only gradually.

VB will also provide the long-missing constructors for its classes, polymorphic methods with type signatures and declaration and assignment in a single statement, like

```
Dim a as Integer = 10
```

Probably not an earthshaking addition, given the awkward syntax.

Finally, VB will introduce threads at long last, so you can write programs that have several active processes at once. This should eventually help develop programs that don't hang during communication or data handling, although you can do this now in simple ways.

We have to recognize that VB is not a completely flexible language when it comes to UI construction, however powerful its visual builder is. It is pretty much impossible to write a program that creates and arranges visual controls programmatically without the UI builder. And I doubt that you will be able to subclass the actual visual controls to make new ones, since this has never been possible in any way in VB. Implementing the Builder pattern, which puts together a set of visual objects based on the data is quite difficult in the general case. So no matter how many new features VB adds this fundamental weakness seems to persist.

Much of the current hype about the new Visual Studio and the VB that comes with it seems to be in the area of Web Forms. Web Forms are yet another attempt by Microsoft to build a set of applications that lock you into IIS (and IE). Web Forms are a kind of advanced Active Server Page that you can create using VB's GUI designer and have reside on your IIS web server. One of the difficult things about Web Forms, which actually exist in some incarnation in VB6 is that they are accompanied by a plethora of utterly impenetrable prose. I've read several articles and explanations of these dinguses in several places, and find that I am as puzzled as ever. One place, they say that they are purely server side programs that generate pure HTML, and in another place they excitedly advance the new interactive event system that allows much greater sophistication in the development of interactions between client and server. Oh boy! More round trips! Web Forms and their cousin (?) WebClass objects are enmeshed in an alphabet soup of Microsoft terminology and at the very least seem to be a closed system rather than an open ended development environment.

Web forms seem to be designed to be used at least partially in conjunction with Visual Interdev, one of the greatest pieces of shelf-ware ever written. I have never seen or tried to use such a completely impenetrable product, and most of the people I talk with share my perplexity.

But in conclusion, I think that the implications of VB7 are really kind of interesting. The inheritance model will finally make it possible for VB programmers to write real OO programs, and may even clean up the nonsense Microsoft has written over the years in which they intentionally confuse interfaces and inheritance to try to hide the fact they VB did not actually support inheritance. VB is a Windows-only programming language that lets you write really nice Windows programs quite easily. If people can write better OO programs as well this only can improve code quality.

And is it a challenge to Java? Well, yes it is, if you regard Java primarily as a client application development language. While I think it is pretty good for that, this hasn't been its focus. And as far as VB being a good client-server development system? Don't hold your breath.

C Sharp or Be Java?

Now C-sharp is a much more sophisticated beast. Unlike the scattered hints about VB7's capabilities, Microsoft has published a 260-page language specification document for C# that leaves little unexplained. Of course C# is just the latent instantiation of internal work Microsoft has had going on for years on a "better C++ language." At one time this language project was also called "Cool."

Missing from this document is anything on graphical aspects of program design. While some of the press releases indicate that the C# system will include a GUI builder and will interface with the dread WebClass objects, this has not yet been documented anywhere.

If you go right down the feature list for C#, you will see a one to one correspondence with Java features. Both languages use more or less the same basic syntax as C++. Table 1 shows significant features that C# and Java have in common.

C#	Java
`namespace, using`	`package, import`
Automatic memory management	Garbage collection
Inheritance (uses :-symbol)	`extends`
`interface (uses :-symbol))`	`interface, implements`
`public, private, protected, internal and protected internal`	`public, private, protected`
`try, catch, finally`	`try, catch, finally`
`lock`	`synchronized`
`const`	`final`
`sealed class`	`final class`

Table 1 – Common features between C# and Java

In general, you can create classes in C# with public and private methods and variables and utilize inheritance, interfaces and most OO programming techniques. But, C# does has some features that I think make it a weaker and more dangerous language. While basic C# does not use pointers, you can create *unsafe* blocks where pointers are permitted. In other words, you have to ask for the rope to hang yourself with. This is analogous to giving a lecture on the mechanics of safe sex without explaining why it is so important to your health.

C# not only has classes, it has structs, that hoary C construct that can dig you into your own grave. Structs lead to a program providing public knowledge of members, like making all of the variables and methods of a class public. Structs are a crutch for programmers who don't really want to write OO code, and can infect a good program with bad code, like the Happy virus would.

C# also supports the goto statement. Anyone knows this is anathema to any kind of program structure, and many believe it should have been replaced years ago by the ComeFrom statement, which is just as clear in practice. I haven't written a goto in any language in almost 20 years and I don't plan to start any time soon.

Passing Parameters to Methods

In Java everything is passed by value, although in the case of objects, it is the value of the object reference. This means that if you pass a primitive type into a class method, the method can change the value of that parameter without changing the value in the calling code. In C#, you can pass a value into a method by value, or by reference. C# also defines an *out* modifier that says that a parameters is for output.

```
public int getStuff(int a, ref int b, out float c) {
```

In the above method, you pass *a* in by value and *b* by reference, and *c* as an output parameter. The method getStuff can return one int to the left of the equals sign, a float in parameter *c*, and optionally one int in parameter *b*. Finally, you can create a method with a *params* parameter as the last argument. A params parameter is an array that you can use to return a whole passel of values. Convenient, perhaps, but hardly readable or obvious.

In Java we have primitive types such as int, float and long and the corresponding objects Integer, Float and Long, which are object wrappers for these simple types. In C#, any type can be treated as an object simply through the syntax you use. The process of converting a primitive type to an object is referred to as Boxing and Unboxing. What would Max Schmeling or George Foreman think of that? The int is down for the count!

C# introduces a new feature they call *Indexers*. As I understand it, these allow you to add an index to an instance of a class and return a value from an array within the class. The general idea is to have a class like

```
Barray bta = new Barray();
BitFlag = bta[3];          //get flag value using Indexer
```

This violates encapsulation big time! You must know there is an array inside the class and you have to know how to ask for data from that array. Why bother with having a class? I don't like this one at all.

Delegating Upward

C# also introduces a Delegate type, which on close reading seems to be a function pointer to a method of some class. Since a Delegate loses its type as far as I can tell, this gives you an untype-checked access to a method in some class instance. Offhand, I can't see why I would ever want to do this in a good OO program.

More Types in C#

In addition to the usual types of int, float, long and bool, C# adds uint, ulong and decimal. The unsigned types could be helpful but the decimal class is easily handled in Java using support classes like BigDecimal. C# allows you to create multidimensional arrays both as arrays of arrays:

```
float x[][] = new float[10][20];
```

or

```
float x[,] = new float[10,20];
```

This may be comforting to non-C programmers but it is scarcely a significant innovation.

I didn't find any sort of Enumeration or Iterator type in the specification, but they did add a foreach looping statement to run through an array of objects. This is apparently a carry over of the For Each statement in Visual Basic, and has no real advantage other than hiding the index variable. The documentation suggests that an Enumerator is used under the covers, but if it is not accessible to the user, it is rather less useful.

C# does add an enum type, giving you a way to have a named series of constants. I haven't missed this much since leaving C for Java, since Enumerations and good OO styles make the enum rather less necessary.

Things that are Missing from C#

First, we have to realize that C# is a proposal for a language. While there is a real specification document, it is far from complete. The purpose of this language and the document to to show that Microsoft plans to submit this document to the ECMA (European Computer Manufacturer's Association) as a proposed standard. This is, of course, a political move more than a technical one, intended to point out that Microsoft is somehow more open than Sun, who withdrew Java from the ECMA standardization process. I can't imagine Microsoft submitting C# to a Java Community process, however.

Still I find it surprising that threads are not mentioned at all in the language document. Now, Java has a wide variety of supporting packages like java.util, java.sql and javax.swing which are nominally part of the Java 2 language spec, but a bit separated from the actual base java.lang package. There is no such variety of supporting classes proposed in C#. So there are no threads, no Vectors or other collections, no database connection, no I/O or file support, and no GUI or printing specified in this document, even though some press release described C# has having a "GUI builder."

I also find it surprising that while you can throw exceptions, there is no "throws" modifier to declare that a method will throw an exception, and force compile time checking of this fact.

Finally, there is no indication that Microsoft intends for this to be a cross-platform language. And if it only runs on Windows, how is it different from what you can already do with C++ class libraries and Microsoft Foundation Classes?

So Is C# Fish or Foul?

It’s hard to imagine a new language and development environment succeeding now that Java is so successful on both servers and clients. It adds very little new capability and really seems to be just a MeToo NotJava language. It’s not just that this dog won’t hunt. It’s that this hunt resulted in a dog.