Java Compared With .NET

James W. Cooper

You probably have heard more than enough about Microsoft's .NET strategy already, and may or may not be a heavy user of this new system. If you are, you may know that the design of Java certainly influenced the language structure of .NET in significant ways. In fact, I have discovered that I can write almost exactly the same code (and I mean this literally) in both systems.

Let's be clear herethat the entire .NET strategy is still unfolding and even that part that has been completely explained is far more than language design. However, by understanding this language design, you'll gain some valuable insights into where .NET can go.

The two most important languages in the .NET system are C# (pronounced "C-sharp") and VB.NET, the later implementation of Microsoft's wildly successful Visual Basic language. While you may think you can dismiss VB out of hand, you do need to realize that it is probably the most widely used computer language in the world (and this means including all platforms). The thing that is unique about the .NET system is that C# and this latest version of VB are just two sides of the same coin. They use the same basic system libraries and compile to the same intermediate code. This intermediate code is then interpreted at execution time much like the Java byte-codes are interpreted by the Java virtual machine. Within this system, both C# and VB.Net now provide garbage collection much like Java does. While it is apparently possible for this intermediate language code to be compiled at execution time into binary Intel code, this is not the current default model.

It seems to me that this approach can be very valuable to Microsoft in moving to 64-bit Intel computing. They do not need to provide specially compiled versions of code written for .NET. They only need to provide a run-time system that is optimized for that platform, and all .NET code will run unchanged. Of course, this also provides a convenient pathway to other completely different platforms such as Linux and the Macintosh, should they wish to pursue competing in that environment.

The greatest strength (and weakness) of C# and VB.NET is the very easy to use visual builder environment that allows you to make nice-looking Windows applications in just a few clicks and drags. All the code to support these windows components is generated automatically for you. And the entire layout is handled under the covers. Even today, there is no Java development environment that does this as well. Of course, the weakness of this system remains that it is Windows-only and this can be a drawback for cross-platform development projects.

The code that generates the windows is now all within your classes and you can change it or generate windows yourself using the same kind of coding the designer GUI produces. VB and C# do not have layout managers at this time, so all of the layouts are in absolute terms. We can of course argue endlessly as to which way is "better," but within just the Windows environment, fixed layout is generally accepted as adequate.

What is C# Like?

The new C# language has the same overall syntax as C, C++ and Java: it uses braces and has the exact same syntax for all the fundamental language elements. C# is considerably more like Java than C++, in that it has classes with methods and constructors and specifically allows only single inheritance. Like Java, C# also supports interfaces. In most C# programming, you don't use pointers at all, although it is possible to create *unsafe* code blocks where pointers are allowed. Probably most reassuringly, the code you write in C# is so much like Java that you can almost paste your Java code into the Visual Studio.NET development environment and compile it. The only difference is in the capitalization of some of the methods. Here is some C# code for splitting a string at the first space:

```
private void splitName(string name) {
    int i = name.IndexOf (" ");
    if (i > 0 ) {
           frname = name.Substring (0, i).Trim();
           lname = name.Substring (i + 1).Trim();
    }
}
```

By comparison, you would write this same code in Java as

```
private void splitName(String name) {
    int i = name.indexOf (" ");
    if (i > 0) {
        first = name.substring(0, i).trim();
        last = name.substring(i + 1).trim();
    }
}
```

In general, the method names are much the same in the common system level classes where you end up writing most of your code. The GUI and file I/O classes are rather different, but since the GUI code is mostly generated for you and since you tend to encapsulate file code in convenient ways, the differences are quite small indeed.

Inheritance in C#

C# has the same single inheritance structure as Java does, although the syntax is slightly more turgid and C++ like. For example, to create a class derived from SwimData, you write

```
public class SexSwimData:SwimData
```

and to pass data to the base class's constructor, you write a derived class constructor that refers to the base class using reserved *base* keyword.

```
public SexSwimData(string filename):base(filename){}
```

I think that the Java approach is a bit easier to read:

```
public class SexSwimData extends SwimData {
      public SexSwimData(String filename) {
            super(filename);
      }
```

C# also allows you to create interfaces rather like Java:

```
public interface MultiChoice      {
       ArrayList getSelected();
       void clear();
       Panel getWindow();
}
```

However, the C# syntax does not make it easy to distinguish inheritance from implementation, as it looks just the same. Here we are creating a class called ListChoice that *implements* the above MultiChoice interface.

```
public class ListChoice:MultiChoice {
```

I think this could be a bit confusing.

Overriding Base Class Methods

C# follows the C++ custom of requiring you to declare that a method can be overridden in derived classes by declaring the base method as *virtual*. Here we make a draw method overridable:

```
public virtual void draw(Graphics g) {
  g.DrawRectangle (rpen, x, y, w, h);
}
```

Then, in the derived class, you must specify that you are overriding that method using the *override* keyword:

```
public override void draw(Graphics g) {
       base.draw (g);      //draw one rectangle
       g.DrawRectangle (rdPen, x +5, y+5, w, h);
}
```

You can also call the base method using the *base* keyword as we show here.

If you don't want to or can't make the base class method virtual, you can use the *new* keyword to create a method which replaces all methods of that name and any signature in the base class with a new one.

```
public new void draw(Graphics g) {
       g.DrawRectangle (rpen, x, y, w, h);
       g.DrawRectangle (rdPen, x +5, y+5, w, h);
}
```

In this case, you no longer can call the parent method in the base class.

VB.NET works in exactly the same way, except that the keywords are much more awkward:

```
Public Overridable Sub draw(ByVal g As Graphics)
     g.DrawRectangle(rpen, x, y, w, h)
End Sub
```

and

```
Public Overrides Sub draw(ByVal g As Graphics)
     MyBase.draw(g)
     g.DrawRectangle(redPen, x + 4, y + 4, w, h)
End Sub
```

To replace a base method instead of overriding it, you use the *Shadows* keyword inVB.NET instead of the *new* keyword used in C#.

Exceptions

C# provides a wide array of standard exceptions that the system may through when errors occur. These include NullReferenceException, DivideByZeroException, FileNotFoundException, and so forth. The try - catch syntax is quite analogous to that in Java

```
try {
      ts = new StreamReader (fileName);
      opened=true;
}
catch(FileNotFoundException e ) {
      Console.WriteLine (e.Message );
}
```

One thing that I find a major disadvantage is that C# does not have a *throws* keyword. There is no way to indicate that a method can throw an exception and that you must prepare your code for that eventuality. If some class you call throws one, you find out when it happens, or by reading the code. So there is no way for the compiler to enforce the need to catch exceptions.

Comparing VB.NET

Now the VB.NET system is also supported by the Visual Studio.NET development environment. You can choose to build either C# or VB programs (or C++). So, the same GUI designer is used for both, but different language specific code is generated. This is the first version of VB to support inheritance, although VB has awkwardly supported interfaces in the past two versions. Because the libraries that this new version of VB interact with are the same ones C# uses, you now have a bunch of new, preferred class methods to use in VB.

For example, to split a string in VB, we might write

```
Private Sub splitName(nm As String)
 Dim i As Integer = nm.IndexOf(" ")
 If i > 0 Then
     Frname = nm.Substring(0, i).Trim()
     Lname = nm.Substring(i + 1).Trim()
 End If
```

Not only is this much the same as C#, it is still much the same as Java. It is actually not entirely unreasonable to convert a Java program to VB.NET by pasting the code into the UI designer and editing it to be correct. In fact, I have done this, and converted all of my Java Design Patterns examples to VB.NET in a very short time. Converting them to C# is even easier.

Inheritance in VB.NET is syntactically clunky, but the development environment manages a lot of it for you. You can create a derived class using the Inherits keyword:

```
Public Class Stocks
    Inherits Equities
```

and the Stocks class will inherit all the methods of the Equities class. Likewise, you can define an interface:

```
Public Interface MultiChoice
    Function getSelected() As ArrayList
```

```
    Sub clear()             'clear all selected
    Function getWindow() As Panel
End Interface
```

And specify classes that implement that interface rather more clearly than you do in C#:

```
Public Class ListChoice
    Implements MultiChoice
```

However, VB forces you to add some awkward syntax to every method that implements this interface:

```
Public Sub clear() Implements MultiChoice.clear
        lst.Items.clear()
    End Sub
```

but the development environment will generate this boilerplate for you.

VB.Net, of course, also supports the same sort of file manipulations and exceptions that C# does, since it is effectively the same language. This is great for C# developers, but does represent a sea change for earlier generations of VB programmers who now have to learn a great number of new method names and some changes in syntax.

Summary

As you can see, the .NET languages bear a lot of relationship to Java, and one might argue that Microsoft's strategy is to use this similarity to tempt Java programmers to move to their system. For certain classes of Windows applications, this clearly might make sense. However, the contrary is also true. Java is now a successful, widely used, multi-platform language, and anyone who learns the .NET languages can now move out of the fold as easily as they could move in. The .NET system may also turn out to be a great training ground for moving VB-style programmers to Java!

James W. Cooper is the author of 14 books. His book *Java Design Patterns: a Tutorial* was published by Addison-Wesley in January, 2000, and his new book *Design Patterns in VB6 and VB.Net* was published in November of 2001.