Philosophical discussion on when to mark a method as virtual

devblogs.microsoft.com/oldnewthing/20060913-04

September 13, 2006

Raymond Chen

When should you mark a method as virtual? This question has both a technical and a philosophical aspect. The technical aspect is well-understood: You mark a method as virtual if you want calls to the method to be invoked on the run-time type of the invoked object rather than on the compile-time type of the invoking reference. But there is a heavy philosophical aspect to this question as well. Some people argue that every method should be virtual so that derived classes can override the behavior of the base class in ways the base class may not have anticipated. This grants you maximum flexibility, avoiding having to redesign the class in the future. I happen to believe that the cost of this extensibility is badly underestimated. Once you mark a method as virtual, your life becomes much more complicated. Every virtual method is an extensibility point, which means that you have to assume the worst. You can't hold any locks when calling a virtual method (else you may deadlock with the derived class). You have to be prepared for all sorts of reentrancy because the derived class may have decided to call `MessageBox` in its implementation. You have to revalidate your state after the virtual method returns because the derived class may have done "crazy" things inside that virtual method including possibly destroying the object itself! (Life gets very interesting when control returns to the base class with an invalid "this" pointer.) It takes only a few seconds to type the word "virtual" but you need to know what you just signed yourself up for. I'm not saying that virtual methods are bad; I'm saying that you need to understand what you are doing. You don't say "Gosh, somebody might want to override this, so I'll make it virtual." You have to say, "Gosh, I'm creating an extensibility point here that needs to be designed and tested."

Of course, if your class is private, then you are the only person who can derive from the class, and therefore you can impose rules on what the derived classes are permitted to do if they override a method, and you have the power to enforce said rules on those derived classes since you're the one who wrote them anyway. But if you allow others to derive from your class, then you have entered the world of extensibility and your life has just gotten a lot more complicated.

Raymond Chen

Follow