

# Re: a case for multiple inheritance

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-06/msg03420.html>

---

- *From:* Tom Spink <[tspink@xxxxxxxxxx](mailto:tspink@xxxxxxxxxx)>
  - *Date:* Thu, 21 Jun 2007 07:01:55 +0100
- 

John wrote:

Tom Spink wrote:

1) Create a generic ProtectedList<T> class which inherits from List<T> and overrides the write functions (using the new modifier) to change the access level from public to protected.

Mhm. FYI The 'new' modifier will define a method as hiding a derived one,

Sorry, I didn't complete the description, the 'new protected' modified functions would delegate to the base implementation. I would assume that the compiler would optimize it so that the double function call would be reduced to a single function call resulting in no performance hit.

however, take a look at this:

```
public class A
{
    public void Foo ()
    {
        Console.WriteLine("Foo in A");
    }
}
```

```
public class B : A
{
    private new void Foo ()
    {
        Console.WriteLine("Foo in B");
    }
}
```

## Re: a case for multiple inheritance

```
}  
}  
  
public class E  
{  
    public static void Main ()  
    {  
        B b = new B();  
        b.Foo();  
    }  
}
```

The output of this will be "Foo in A", because the accessible method from A is called. If you change the scope of 'Foo' in B to public, the output will be "Foo in B" as is expected. And, even more interestingly, if you cast b to A and call 'Foo':

```
((A)b).Foo();
```

You'll get A's implementation of 'Foo'. So, as you can see, using the new keyword will not give you the desired effect here.

The automatic fallback to the public base implementation, after specifically overriding it, is very disturbing to me. The cast not much of a surprise, assuming you are allowed to perform that cast, I would expect that behavior.

2) Inherit both specialized List<T> by the high-level class

Can you explain this a bit more? I'm not