Why to use the sealed keyword on classes

interface IInterface1 {}

class Class1 {}

class Program

{

static void Main(string[] args)

{

*//Class1 does not implement IInterface1*

Class1 instanceOfClass1 = new Class1();

*//However this cast does not leads to compilation error*

IInterface1 someImplementer = (IInterface1)instanceOfClass1;

}

}

Now let’s make Class1 sealed. Now the compiler will see that Class1 can be only Class1 “itself” (and its base classes if applicable) and it does not implement interface IInterface1.

Hide   Copy Code

internal interface IInterface1 {}

sealed class Class1 {}

class Program

{

static void Main(string[] args)

{

*//Class1 does not implement IInterface1*

Class1 instanceOfClass1 = new Class1();

*//However this cast does not leads to compilation error*

IInterface1 someImplementer = (IInterface1)instanceOfClass1;

}

}

The following compilation error will occur:

Hide   Copy Code

Cannot convert type 'Class1' to 'IInterface1'

USES

|  |  |
| --- | --- |
| 24down voteaccepted | 1) On a class that implements security features, so that the original object cannot be "impersonated".  2) More generally, i recently exchanged with a person at Microsoft, who told me they tried to limit the inheritance to the places where it really made full sense, because it becomes expensive performance-wise if left untreated. The sealed keyword tells the CLR that there is no class further down to look for methods, and that speeds things up.  In most perfomance-enhancing tools on the market nowadays, you will find a checkbox that will seal all your classes that aren't inherited. Be careful though, because if you want to allow plugins or assembly discovery through MEF, you will run into problems. |