What is the difference between the override and new keywords in C#?

Asked 10 years, 3 months ago Active 8 months ago Viewed 39k times

82

What is the difference between the `override` and `new` keywords in C# when defining methods in class hierarchies?

c# .net oop

24

Share Edit Follow

edited Jun 28 '12 at 10:33
Community Bot
1 ● 1

asked Jul 4 '11 at 21:27

jaywayco
4,786 ● 5 ● 23 ● 40

6 Answers

Active Oldest Votes

111

The following page summarizes your question very nicely.

[Knowing When to Use Override and New Keywords]

Summary

Override: When a method of a base class is overridden in a derived class, the version in the derived class is used, even if the calling code didn't "know" that the object was an instance of the derived class.

New: If you use the new keyword instead of override, the method in the derived class doesn't override the method in the base class, it merely hides it.

If you don't specify either new or overrides, the resulting output is the same as if you specified new, but you'll also get a compiler warning (as you may not be aware that you're hiding a method in the base class method, or indeed you may have wanted to override it, and merely forgot to include the keyword).

Override: used with virtual/abstract/override type of method in base class

New: when base class has not declared method as virtual/abstract/override

Share Edit Follow

edited Jun 20 '20 at 9:12

answered Jul 4 '11 at 21:28

Community Bot
1 ● 1

Brian Graham
12.4k ● 12 ● 55 ● 96

11 Can you include the salient information here as well please. It helps keep the information here and guards against link rot (unlikely with an MSDN blog, but you never know). – ChrisF ♦ Jul 4 '11 at 21:29

4 lol "unlikely" I get an redirect followed by "Access Denied You do not have permission to view/download this item." – tobsen Apr 28 '13 at 15:45

82

`new` will shadow the method with a completely new method (which may or may not have the same signature) instead of overriding it (in which case the new method must have the same signature), meaning that polymorphism won't work. For example, you have these classes:

```
class A {
    public virtual int Hello() {
        return 1;
    }
}

class B : A {
    new public int Hello(object newParam) {
        return 2;
    }
}

class C : A {
    public override int Hello() {
        return 3;
    }
}
```

If you do this:

```
A objectA;
B objectB = new B();
C objectC = new C();

Console.WriteLine(objectB.Hello(null)); // 2
Console.WriteLine(objectC.Hello()); // 3

objectA = objectB;

Console.WriteLine(objectA.Hello()); // 1

objectA = objectC;

Console.WriteLine(objectA.Hello()); // 3
```

Since you can define new method signatures with `new`, it's impossible for the compiler to know that the instance of `A` is actually an instance of `B` and the new

method `B` defines should be available. `new` can be used when the parent object's method, property, field or event is not declared with `virtual`, and because of the lack of `virtual` the compiler won't "look up" the inherited method. With `virtual` and `override`, however, it works.

I would strongly recommend you avoid `new`; at best, it's confusing, because you're defining a method with a name that could be recognized as something else, and at worst, it can hide mistakes, introduce seemingly impossible bugs, and make extending functionality difficult.

Share Edit Follow

edited Jun 30 '14 at 19:41

answered Jul 4 '11 at 21:37

Ry- ♦

203k ● 52 ● 419 ● 427

15 This should be the accepted answer. It actually explains, with examples, what the differences are and the implications. The accepted answer is pretty vague. – theyetiman Aug 28 '15 at 15:55

1 something similar happens when using an interface (as in B extends A, A implements interface with `Hello()`, assign B to variable with interface type). Would be good to clarify that also in your answer, as that was the situation I was looking for. – ahong Jun 2 '20 at 7:43

10

Looks like an old question, let me try a different answer:

1. `new` : as the name says, it is a new member in the family of inheritance hierarchy and this will be used as base member for further down the chain (if marked as virtual).
2. `override` : It means I don't accept my parent class' member implementation and I will do differently.

Share Edit Follow

edited May 29 '15 at 8:08

Vegard Larsen

11.8k ● 12 ● 55 ● 99

answered Apr 8 '12 at 15:57

Dhananjay

3,397 ● 2 ● 20 ● 19

4

Consider the following class hierarchy:

```
using System;

namespace ConsoleApp
{
     public static class Program
     {
          public static void Main(string[] args)
          {
```

```
            Overrider overrider = new Overrider();
            Base base1 = overrider;
            overrider.Foo();
            base1.Foo();

            Hider hider = new Hider();
            Base base2 = hider;
            hider.Foo();
            base2.Foo();
        }
    }

    public class Base
    {
        public virtual void Foo()
        {
            Console.WriteLine("Base      => Foo");
        }
    }

    public class Overrider : Base
    {
        public override void Foo()
        {
            Console.WriteLine("Overrider => Foo");
        }
    }

    public class Hider : Base
    {
        public new void Foo()
        {
            Console.WriteLine("Hider     => Foo");
        }
    }
}
```

Output of above codes must be:

```
Overrider => Foo
Overrider => Foo

Hider     => Foo
Base      => Foo
```

> - A subclass `overrides` a virtual method by applying the `override modifier`:
> - If you want to `hide` a member deliberately, in which case you can apply the `new modifier` to the member in the subclass. `The new modifier does nothing more than suppress the compiler warning that would otherwise result`

Share Edit Follow

answered Feb 8 '18 at 16:23

Sina Lotfi


2,255 ● 1 ● 17 ● 27

1 good example, I think it would be more readable if you split up the `Program` class and other classes into two separate code blocks, with the output inline with the method invocation as a comment, e.g. `base2.Foo(); // Base => Foo`. I would also leave out the namespace and import, but if you want to provide runnable code, then just link to a dotnetfiddle.net – ahong Jun 2 '20 at 8:12

2

`override` lets you override a virtual method in a base class so that you can put a different implementation in. `new` will hide a non-virtual method in a base class.

Share Edit Follow

answered Jul 4 '11 at 21:29


Daniel Mann
51.2k ● 12 ● 92 ● 108

this much i knew, however, override will effectively hide a base classes method – jaywayco Jul 4 '11 at 21:32

2 No override does not hide the base class method. Override makes calls to the base class method into calls to the derived class method. In a way with override the method in the derived class is the same method as the one in the base class. Whereas with `new` it's a completely independent method that just happens to have the same name. – CodesInChaos Jul 4 '11 at 21:35

@CodeInChaos Thanks for saying what I wanted to say better than I was able to say it :) – Daniel Mann Jul 4 '11 at 21:37

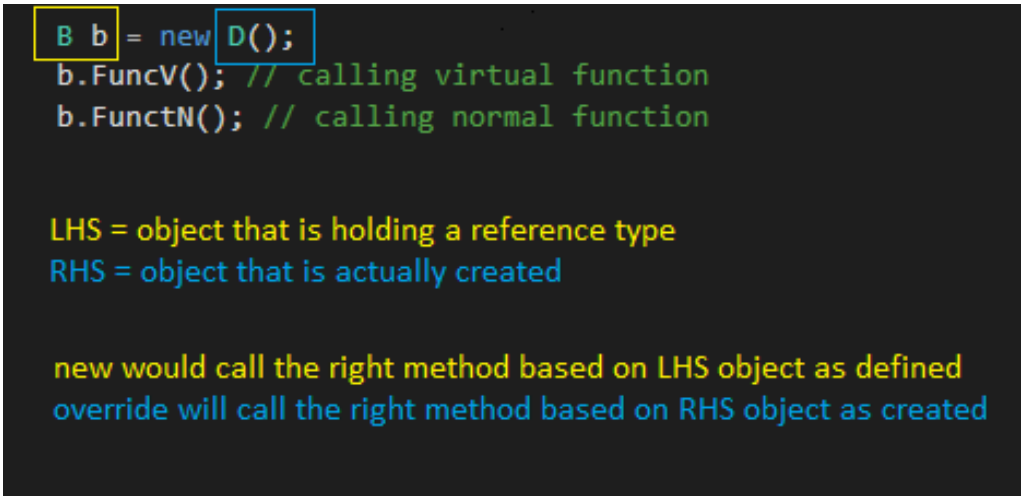
1

The simple difference is that `override` means the method is virtual (it goes in conduction with `virtual` keyword in base class) and `new` simply means it's not virtual, it's a regular override.

So both really are function overrides, one is with virtual characteristics, the other not.

What does mean exactly? It simply means polymorphism will not be in play for `new' methods.

The following image illustration might make this clear.



Note if you don't use `new` keyword, it is still implied but it will generate a warning message.

Share Edit Follow

answered Jan 24 at 7:23

zar

9,649 ● 10 ● 77 ● 147