X

How to invoke parent class method without creating object of it

Asked 9 years, 5 months ago Active 9 years, 5 months ago Viewed 9k times



1

I can invoke the parent class method by using base.virtualParentMethod(). But how do I call the method in parent-parent class without creating an object of it, in the following scenario.



*



```
class A
   {
       public virtual void virtualParentMethod()
       {
           Console.WriteLine("A");
       }
   class B : A
       public override void virtualParentMethod()
       {
           Console.WriteLine("B");
       }
   }
   class C : B
       public override void virtualParentMethod()
           //base.virtualParentMethod();
           //This is where I want to invoke the method of A
           //So that out Will be : A
       }
   }
```

```
c# .net oop
```

Share Edit Follow



If you need to do this, the odds are you need to refactor this. You seem to have a mismatch between your inheritance model and your desired behaviour - Pete Apr 16 '12 at 12:13

duplicate - stackoverflow.com/questions/438939/... - scibuff Apr 16 '12 at 12:14

If you really want to do this, perhaps you should consider changing your design... – Torbjörn Kalin Apr 16 '12 at 12:15





If you need some parent functionality in not direct children of parent, then you should move that functionality to separate method:

4



43

```
class A
{
    public virtual void VirtualParentMethod()
        Foo();
    }
    protected void Foo()
        Console.WriteLine("A");
}
class B : A
    public override void VirtualParentMethod()
        Console.WriteLine("B");
    }
}
class C : B
    public override void VirtualParentMethod()
        Foo();
    }
}
```

UPDATE

Also consider:

- inheriting c directly from A
- changing inheritance order, i.e. B from C
- · extracting this behavior to other object

Share Edit Follow

edited Apr 16 '12 at 12:25

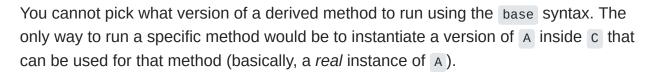
answered Apr 16 '12 at 12:18

Sergev Berezovskiv





2





However I personally wouldn't do this. This is indicative of a design issue with your inheritance chain.

If you need to ensure that c runs A, why not inherit A from C - C : A.

Share Edit Follow

edited Apr 16 '12 at 12:20

answered Apr 16 '12 at 12:14





You simply can not, cause that is an essence of <u>Virtual Method Table</u>, so <u>virtual</u> keyword and its override. Invokation method's address will be lookuped from corresponding *real* type Virtual Methods Table.



Share Edit Follow

Latt 1 offerv





Tigran 59.8k • 8 • 81 • 118



0

You can't. using base.MethodName will always call the closest implementation up the hierarchy and there's no way you can by pass this without explicitly declaring a method that can be called



A)

it's also worth noting that if you find yourself in a situation where you wish to do this odds are you have a design flaw.

Share Edit Follow

answered Apr 16 '12 at 12:17



Rune FS 20.8k • 6 • 57 • 93

0

You've already created an instance of A, by creating an instance of C (as C : B, B : A), so there is a way you can do this with a small modification. Now, without indicating whether or not this is a good idea or not;



If you use the 'new' keyword instead of override on your derived method implementations, then you can call the base implementation of a method directly, bypassing the intermediate class implementation, by casting the object to the type who's implementation you want to use, like so;

```
class A
{
    public virtual void virtualParentMethod()
    {
        Console.WriteLine("A");
    }
} class B : A
{
    public new void virtualParentMethod()
    {
        Console.WriteLine("B");
```

Note that if you make this change, you've introduced a behavioural change to any callers of the method, depending on how they refer to the object. And if you try to cast in this way in the existing C.virtualParentMethod implementation (declared 'override'), you are really just calling the method itself and will get in an infinite loop.

Or, you could just reconsider your class design. :-)

Share Edit Follow

answered Apr 18 '12 at 23:51

RJ Lohan

6,317 • 3 • 32 • 53