Calling the base constructor in C#

Ask Question

1291

157

If I inherit from a base class and want to pass something from the constructor of the inherited class to the constructor of the base class, how do I do that?

For example,

If I inherit from the Exception class I want to do something like this:

```
class MyExceptionClass : Exception
{
     public MyExceptionClass(string message, string extraInfo)
     {
         //This is where it's all falling apart
         base(message);
     }
}
```

Basically what I want is to be able to pass the string message to the base Exception class.

c# inheritance constructor

edited Apr 5 at 11:32
Ola Ström
53 8

asked Aug 15 '08 at 7:39
lomaxx

57.2k 54 131 172

24 Its also worth noting you can chain constructors in your current class by substituting `this` for `base` . – Quibblesome Aug 15 '08 at 13:30

10 Answers

1623

Modify your constructor to the following so that it calls the base class constructor properly:

```
public class MyExceptionClass : Exception
{
    public MyExceptionClass(string message, string extrainfo) : bas
    {
        //other stuff here
    }
}
```

Note that a constructor is not something that you can call anytime within a method. That's the reason you're getting errors in your call in the constructor body.

edited Aug 15 '08 at 7:48

answered Aug 15 '08 at 7:40

Jon Limjap
79.9k 14 91 146

36 I think you may have missed the point. The problem was about calling a base constructor midway through the overriden constructor. Perhaps the data-type of the base constructor is not the same or you want to do some data moulding before passing it down the chain. How would you accomplish such a feat? – Marchy Feb 16 '09 at 21:03

187 If you need to call the base constructor in the middle of the override, then extract it to an actual method on the base class that you can call explicitly. The assumption with base constructors is that they're absolutely necessary to safely create an object, so the base will be called first, always. – Jon Limjap Feb 17 '09 at 10:46

32 It *is* just a method you can call any time, IL-wise. C# just happens to put extra restrictions on top of this. – Roman Starkov Apr 17 '11 at 3:03

14 It is worth noting that the `base` constructor is called *before* the method block is accessed. msdn.microsoft.com/en-us/library/ms173115.aspx – John Weisz Dec 8 '15 at 16:27

18 It is not a good design if you need to call the base class constructor midway during your constructor. The idea of a constructor is that it does all the work needed to do its task. This has the effect that when your derived constructor starts, the base class is already fully initialized and the derived class is free to call any base class function. If your design is such that you want to do something half way your constructor, then apparently this is not initializing the base class ans thus should not be in the constructor of the base class but in a separate, possibly protected function – Harald Coppoolse Dec 16 '15 at 7:28

6

As per some of the other answers listed here, you can pass parameters into the base class constructor. It is advised to call your base class constructor at the beginning of the constructor for your inherited class.

```
public class MyException : Exception
{
    public MyException(string message, string extraInfo) : base(mess
    {
    }
}
```

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams
Q&A for work

Learn More

I note that in your example you never made use of the `extraInfo` parameter, so I assumed you might want to concatenate the `extraInfo` string parameter to the `Message` property of your exception (it seems that this is being ignored in the accepted answer and the code in your question).

This is simply achieved by invoking the base class constructor, and then updating the Message property with the extra info.

```
public class MyException: Exception
{
    public MyException(string message, string extraInfo) : base($"{m
{extraInfo}")
    {
    }
}
```

edited Mar 19 at 6:31

answered Mar 14 '17 at 12:38
Daffy Punk
1,371 1 15 25

this.Message is read only, so solution can't work. – Andy Middleditch Mar 18 at 14:42

@AndyMiddleditch No, its not. The setter is just private. docs.microsoft.com/en-us/dotnet/api/… – Daffy Punk Mar 18 at 16:01

@AndyMiddleditch But you are correct that the previous version of my example would not compile due to the fact that the message attribute has a private setter. I updated my example to work around this (it will now compile). Thanks for the heads-up. – Daffy Punk Mar 19 at 6:33

13

You can also do a conditional check with parameters in the constructor, which allows some flexibility.

```
public MyClass(object myObject=null): base(myObject ?? new myOtherOb
{
}
```

or

```
public MyClass(object myObject=null): base(myObject==null ? new myOt
myObject)
{
}
```

edited Jun 11 '18 at 10:47

answered May 27 '16 at 21:27

C0r3yh
1,579 19 24

1 Don't you need to remove the words "class" from your example since these are constructors... – MarzSocks Sep 27 '16 at 4:50

26

It is true use the `base` (something) to call the base class constructor, but in case of overloading use the `this` keyword

```
public ClassName() : this(par1,par2)
{
// do not call the constructor it is called in the this.
// the base key- word is used to call a inherited constructor
}

// Hint used overload as often as needed do not write the same code
```

edited Apr 9 '18 at 11:46

shA.t
13.1k 4 38 71

answered Nov 11 '13 at 11:32

Janus Pedersen
287 3 3

6 I see what you are trying to explain, and you are right. If you have two constructors in one class, you can reference one from the other by using the "this" keyword similarly to how you use "base" when calling the inherited constructor. However, this isn't what the OP asked for so this isn't really the place to add this. – IAmTimCorey Dec 4 '13 at 14:35

7

```
public class MyException : Exception
{
    public MyException() { }
    public MyException(string msg) : base(msg) { }
    public MyException(string msg, Exception inner) : base(msg, inne
}
```

edited Mar 22 '17 at 13:10

answered Apr 7 '16 at 12:06

Donat Sasin
111 1 2 6

1 this is the best answer because it contains constructor overloads as well. – vibs2006 Jun 10 '18 at 2:56

17

From Framework Design Guidelines and FxCop rules.:

1. Custom Exception should have a name that ends with Exception

```
class MyException : Exception
```

2. Exception should be public

```
public class MyException : Exception
```

3. CA1032: Exception should implements standard constructors.

- A public parameterless constructor.
- A public constructor with one string argument.
- A public constructor with one string and Exception (as it can wrap another Exception).
- A serialization constructor protected if the type is not sealed and private if the type is sealed. Based on MSDN:

```
[Serializable()]
public class MyException : Exception
{
  public MyException()
  {
     // Add any type-specific logic, and supply the default mess
  }

  public MyException(string message): base(message)
  {
     // Add any type-specific logic.
  }
  public MyException(string message, Exception innerException):
     base (message, innerException)
  {
     // Add any type-specific logic for inner exceptions.
  }
  protected MyException(SerializationInfo info,
     StreamingContext context) : base(info, context)
  {
     // Implement type-specific serialization constructor logic.
  }
}
```

or

```
   [Serializable()]
   public sealed class MyException : Exception
   {
     public MyException()
     {
        // Add any type-specific logic, and supply the default mess
     }

     public MyException(string message): base(message)
     {
        // Add any type-specific logic.
     }
     public MyException(string message, Exception innerException):
        base (message, innerException)
     {
        // Add any type-specific logic for inner exceptions.
     }
     private MyException(SerializationInfo info,
        StreamingContext context) : base(info, context)
     {
        // Implement type-specific serialization constructor logic.
     }
   }
```

answered Jan 24 '16 at 7:34

Fab
6,479 1 27 53

Note that you can use **static** methods within the call to the base

461

constructor.

```
class MyExceptionClass : Exception
{
     public MyExceptionClass(string message, string extraInfo) :
         base(ModifyMessage(message, extraInfo))
     {
     }

     private static string ModifyMessage(string message, string extr
     {
         Trace.WriteLine("message was " + message);
         return message.ToLowerInvariant() + Environment.NewLine + e
     }
}
```

edited Aug 31 '15 at 17:17
Abhishek
2,233 1 24 49

answered Apr 28 '10 at 17:34
Axl
5,919 2 20 18

1 The Exception class is so locked down that I do find myself doing this a couple times, but also note it's not something you should do if you can avoid it. – Jonathon Cwik Mar 11 '15 at 19:02

Hi @ChrisS, can I use like ` : base("My default message.")`, how use like this? – Carlos May 15 '15 at 18:32

Sorry, C# newb here. Why do you you call `Trace.WriteLine("message was " + message)` ? – kdbanman Jul 9 '15 at 22:08

4 @kdbanman That just outputs a debug message. No relevant functional purpose. – Nick Whaley Aug 21 '15 at 13:52

3 Great answer. Accepted answer doesn't allow for me to do processing; and the followup comment on a workaround assumes I have access to change the base class; I don't. A factory answer assumes that I can control how the class is instantiated; I can't. Only your answer lets me

modify something before passing it on to base. – The Red Pea May 26 '17 at 21:38

6

```
class Exception
{
     public Exception(string message)
     {
         [...]
     }
}

class MyExceptionClass : Exception
{
     public MyExceptionClass(string message, string extraInfo)
     : base(message)
     {
         [...]
     }
}
```

answered Feb 26 '15 at 4:11

Tutankhamen
2,716 1 22 32

87

If you need to call the base constructor but not right away because your new (derived) class needs to do some data manipulation, the best solution is to resort to factory method. What you need to do is to mark private your derived constructor, then make a static method in your class that will do all the necessary stuff and later call the constructor and return the object.

```
public class MyClass : BaseClass
{
    private MyClass(string someString) : base(someString)
    {
        //your code goes in here
    }

    public static MyClass FactoryMethod(string someString)
    {
        //whatever you want to do with your string before passing it
        return new MyClass(someString);
    }
}
```

edited Apr 7 '13 at 20:52

answered Feb 27 '13 at 2:22

armanali
1,465 14 31

4 This could **potentially** violates the SOLID principles (SRP), because the responsibility of creating the class is encapsulate with whatever other responsibility the class was suppose to take care of. An abstract factory could be used but might add unnecessary complexity to simple code. Of course violation of the SOLIDs is ok if you know the trade off and the toll it is going to put on your architecture (and how to fix any future issues that might arise from your design decision). – Sebastien Jan 10 '17 at 21:22

25

```
public class MyExceptionClass : Exception
{
    public MyExceptionClass(string message,
      Exception innerException): base(message, innerException)
    {
        //other stuff here
    }
}
```

You can pass inner exception to one of the constructors.

answered Dec 4 '09 at 5:03

SnowBEE
650 1 8 14