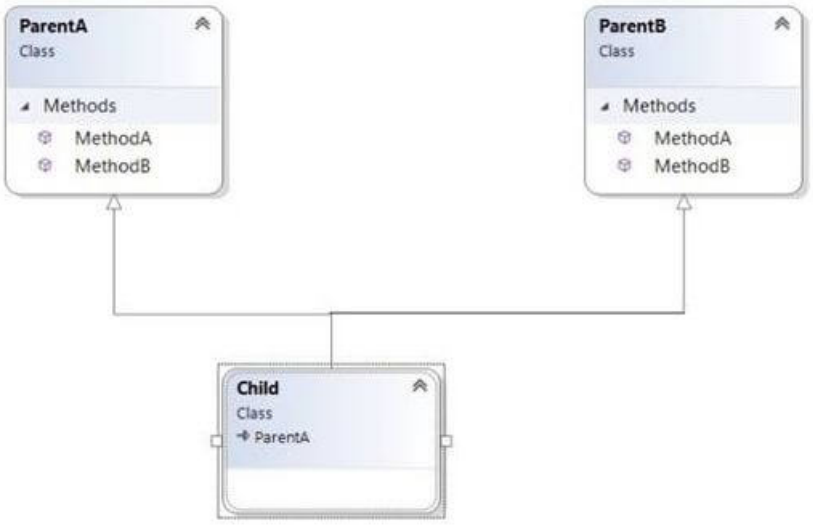
Implement Multiple Inheritance In C#

Atul Sharma Updated date May 28, 2019 40.8k 1 3

After reading the title of this article, the first thing that gets clarified in our head is that C# doesn't allow Multiple Inheritance. However, there are a few languages that allow this. Let us first investigate why C# and Java don't allow multiple Inheritance.

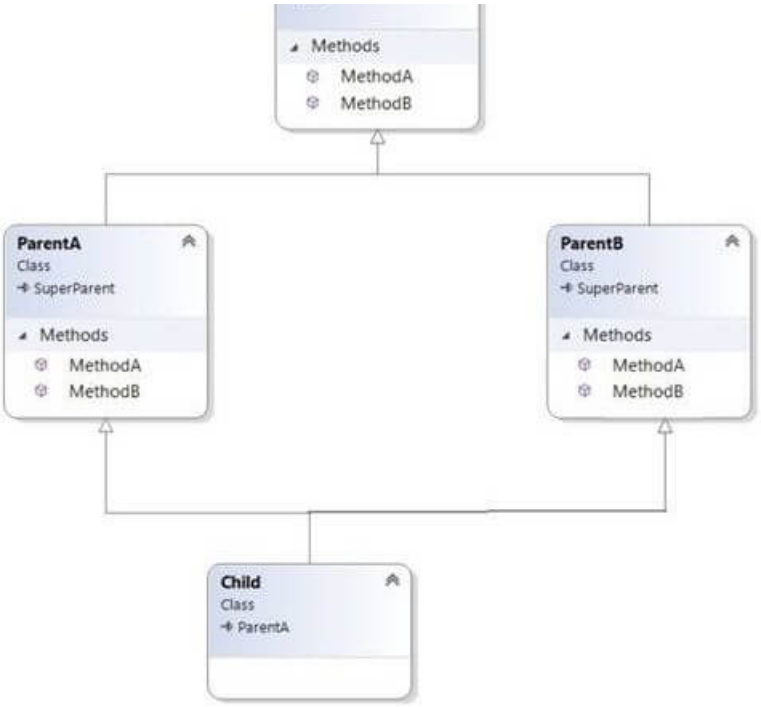
Multiple Inheritance - Why is it not allowed?



Considering the above hypothetical class diagram. We want to have a class Child which inherits Parent classes *ParentA* and *ParentB.* Both parent classes have the same methods, *MethodA* and *MethodB.*

Now, when we instantiate the *Child* class, then calling *MethodA* will confuse the compiler regarding from which class *MethodA* should be called.

The similar case will be observed when both classes (*ParentA* and *ParentB*) inherit a *SuperClass* as shown here.



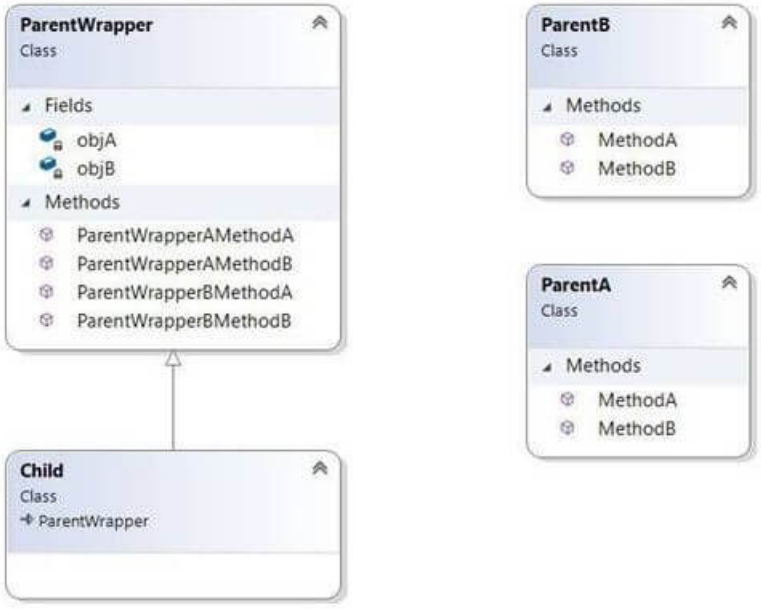
Since this structure resembles a diamond, this problem is famous as the Diamond Problem (See the similarity between the class diagram and diamond shape) and we often hear "Because of the Diamond Problem, languages don't allow multiple inheritance."

Implementing Multiple Inheritance

In real life, we can get into a situation where we need to implement multiple inheritance. So, let us see the workarounds to achieve this.

Approach #1

In this approach, we make a wrapper class, ParentWrapper, and have methods from both of the classes. This is a way to combine the classes.



```
class ParentA
  {
      public void MethodA()
      {
          Console.WriteLine("MethodA from ParentA called");
      }
      public void MethodB()
      {
          Console.WriteLine("MethodB from ParentA called");
      }
  }

  class ParentB
  {
      public void MethodA()
      {
          Console.WriteLine("MethodA from ParentB called");
      }
      public void MethodB()
      {
          Console.WriteLine("MethodB from ParentB called");
      }
  }

  class ParentWrapper
  {
      ParentA objA = new ParentA();
      ParentB objB = new ParentB();
      public void ParentWrapperAMethodA()
      {
          objA.MethodA();
      }
      public void ParentWrapperAMethodB()
      {
          objA.MethodB();
      }
      public void ParentWrapperBMethodA()
      {
          objB.MethodA();
      }
      public void ParentWrapperBMethodB()
      {
          objB.MethodB();
      }
  }

  class Child : ParentWrapper
  {

  }
```

This is how we can call them.

```
Child objChild = new Child();
objChild.ParentWrapperAMethodA();
objChild.ParentWrapperBMethodB();
```

Approach #2

In the previous approach, we see that combining both the classes could be a big headache and here, we have the second approach to implement the same.

In this approach, we have a class ParentB implemented an Interface IParentB (@Line#17). We need to make sure that the interface *IParentB* has all methods defined in ParentB class (From Line# 19-22). Now, our class *Child* will inherit Class *ParentA* and implement Interface *IParentB* (@Line#32). Since class Child is implementing *IParentB* interface, so we will have to implement all the methods of this in *Child* class and we can have it referencing *ParentB* (From Line#35-38).

Code implementation of this approach will be like this.

```
01. class ParentA
02.  {
03.      public void MethodX()
04.      {
05.          Console.WriteLine("MethodX from ParentA called");
06.      }
07.      public void MethodY()
08.      {
09.          Console.WriteLine("MethodY from ParentA called");
10.      }
11.      public void MethodZ()
12.      {
13.          Console.WriteLine("MethodZ from ParentA called");
14.      }
15.  }
16.
17.  class ParentB : IParentB
18.  {
19.      public void MethodA()
20.      {
21.          Console.WriteLine("MethodA from ParentB called");
22.      }
23.
24.  }
25.
26.  interface IParentB
27.  {
28.      void MethodA();
29.
30.  }
31.
32.  class Child : ParentA, IParentB
```

```
        public void MethodA()
        {
            objParentB.MethodA();
        }
    }
```

And here, we can call it.

```
Child objChild2 = new Child();
objChild2.MethodX();
objChild2.MethodA();
```

Output

In both the scenarios, we see this output.

```
First Approach
MethodA from ParentA called
MethodB from ParentB called
Second Approach
MethodX from ParentA called
MethodA from ParentB called
```

When to use which approach?

Approach #1

- It will be good when the class size is small and it doesn't have too many methods.
- We are just provided classes and can NOT modify them at all.
- All constituent classes have the same method name.

Approach # 2,

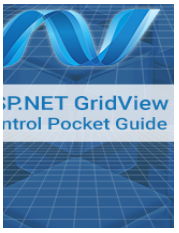
- This approach is best suited when at least one class could be modified.
- When one class has a smaller number of methods than another class.
- When methods are different in all classes.

I hope, with this, I could explain the reason for not having Multiple inheritance. If extremely needed, I have explained the ways to implement this along with the best evaluation of the best approach in both of the scenario.

Source Code to experiment with the concepts explained in this article is available here.

C# | Inheritance | Inheritance In C# | Multiple Inheritance In C#

Next Recommended Reading
Simulating Multiple Inheritance in C#: Part II



Atul Sharma TOP 500

I have been working in Microsoft Technologies Stacks for more than a decade. When I am not writing codes then love to write poems, travel stories and a lot more. I have bundled up all my writings here -

https://taagung.com/

375 547.7k 2

3 1

Type your comment here and press Enter Key (Minimum 10 characters)

Extremely helpful..please explain more when we needed in real time scenario..

Vikas Singh Jun 22, 2019

1913 174 780 0 0 Reply

FEATURED ARTICLES

What Is Web3

What Is Metaverse

Built-In Functions In SQL Server

How To Upgrade to Windows 11

Exploring Subject <T> In Reactive Extensions For .Net

View All

TRENDING UP

01 How to Implement JWT Authentication in Web API Using .Net 6.0, Asp.Net Core

02 JWT Authentication And Authorization In .NET 6.0 With Identity Framework

03 Handling Data Load Types In Staging Database

04 Three Tier Architecture In ASP.NET Core 6 Web API

05 ASP.NET Webform User Registration With Captcha

06 How To Implement Caching In The .NET Core Web API Application

07 An Introduction To Azure DevOps And Its Features

Rockin' The Code World with dotNetDave ft. Daniel Roth Ep. 46

09 How To Use Dependency Injection In .NET Core

10 How To Post Data In ASP.NET Using AJAX Without JSON Form Serializer

View All