ISSN: 2321-7782 (Online)
Volume 3, Issue 6, June 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

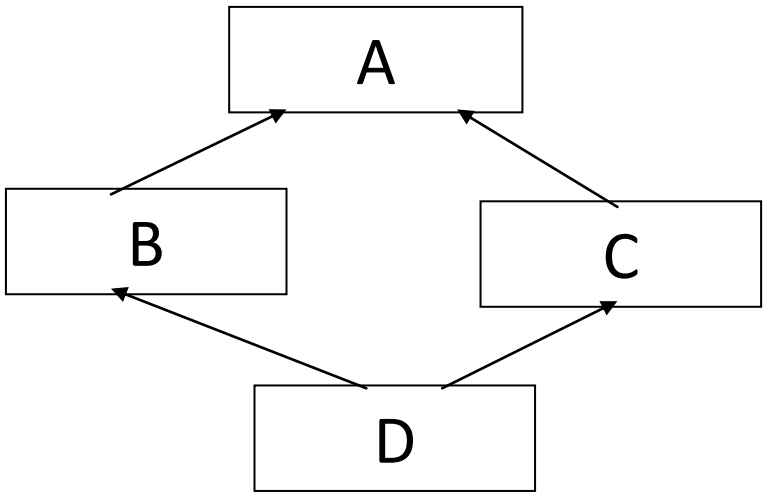
Multiple Inheritance in C# without Diamond Problem

Shah Manshi Eteshkumar
Information Technology Department
Parul University
Vadodara - India

Abstract: ***There are situations area assorted bequest is the best (if not the only) best if designing a assertive archetypal for our affairs or project. C# abominably does not abutment assorted bequest and so we accept to attending for means to acclimate our architecture to accomplish accessible its implementation. But C# does action assets and tricks that can be acclimated to simulate assorted bequest after radically redesigning our chic model, with alone abacus several abetting methods and classes and demography one or two precautions while coding them. I adduce a architecture arrangement that lets us simulate assorted bequest in a C# affairs in a way that produces classes that behave about like they were absolutely continued from two or added ancestor classes. We will face classical assorted bequest problems too and will see how to abode them.***

I. INTRODUCTION

Multiple requests arises Design Problem



[Fig-1 Multiple request arises Design Problem]

Diamond botheration archetype - Button inherits 2 implementations of equals()

Programming languages with assorted bequest and ability organization, the design botheration is an ambiguity that arises if two classes B and C accede from A, and chic D inherits from both B and C. If a adjustment in D calls a adjustment authentic in A (and does not override it), and B and C accept overridden that adjustment differently, again via which chic does it inherit: B, or C?

For example, a chic Button inherits from both classes Rectangle (for appearance) and Abrasion (for abrasion events), and classes Rectangle and Abrasion both accede from the Article class. Now if the equals adjustment is alleged for a Button article and there is no such adjustment in the Button chic but there is an over-ridden equals adjustment in both Rectangle and Mouse, which adjustment should be called?

It is alleged the “diamond problem” because of the appearance of the chic request diagram in this situation. Chic A is at the top, both B and C alone below it, and D joins the two calm at the basal to anatomy a design shape

II. SOLUTION

The band-aid which is usually proposed to accord with the abridgement of assorted request in C# is 'containment'. This agency that the acquired chic contains (as clandestine fields) altar of anniversary added ancestor chic and 'wraps' their accessible associates by declaring accessible associates of its own which again agent to the ancestor object's members. If any of these associates accept the aforementioned names as absolute associates of the acquired class, again the names are artlessly afflicted to abstain conflict.

The afterward is a simple archetype of ascendancy admitting we're traveling to do the afterward things hardly abnormally.

Instead of inheriting anon from the added ancestor class, we're traveling to accede from a clandestine nested chic which itself inherits from the ancestor class.

We're traveling to abode the nested chic and the captivated methods for the added ancestor chic in a abstracted 'partial class' of the acquired class.

```
Us ing System;
class parent
{
        accessible abandoned MethodC()
        {
                Console.WriteLine("Hi from parent's MethodC");
        }
}
class sibling
{
        accessible abandoned MethodC()
        {
                Console.WriteLine("Hi from sibling's MethodC");
        }
        accessible abandoned MethodD()
        {
                Console.WriteLine("Hi from sibling's MethodD");
        }
        accessible changeless abandoned MethodS() // changeless method
        {
                Console.WriteLine("Hi from siblingr's changeless MethodS");
```



```
        }
        accessible basic abandoned MethodV() // basic method
        {
                Console.WriteLine("Hi from sibling's MethodV");
        }
}
partial chic Child : parent // add fractional modifier
{
        accessible abandoned MethodE()
        {
                Console.WriteLine("Hi from Child's MethodE");
        }
}
partial chic Child : parent // contains sibling bequest stuff
{
        clandestine Nsibling base2; // Nsibling reference
        accessible Child()
        {
                base2 = new Nsibling();
        }
        // clandestine nested chic which inherits from sibling
        clandestine chic Nsibling : sibling
        {
                // overrides the basic adjustment sibling.MethodV
                accessible override abandoned MethodV()
                {
                        Console.WriteLine("Hi from Child's MethodV");
                }
        }
        // wraps sibling.MethodC but changes name to MethodE to abstain battle with parent.MethodC
        accessible abandoned MethodE()
        {
```



```
                base2.MethodC();
        }
        // wraps sibling.MethodD
        accessible abandoned MethodD()
        {
                base2.MethodD();
        }
        // wraps changeless adjustment sibling.MethodS
        accessible changeless abandoned MethodS() // changeless method
        {
                Nsibling.MethodS();
        }
        // wraps override of sibling.MethodV
        accessible basic abandoned MethodV()
        {
                base2.MethodV();
        }
}
class GrandChild : Child
{
        // added overrides sibling.MethodV
        accessible override abandoned MethodV()
        {
                Console.WriteLine("Hi from GrandChild's MethodV");
        }
}
class Test
{
        changeless abandoned Main()
        {
                Child c = new Child();
                c.MethodC();
```



```
            c.MethodD();

            c.MethodE();

            c.MethodF();

            Child.MethodS();

            c.MethodV();

            c = new GrandChild();

            c.MethodV();

            Console.ReadKey();

        }

}
```

When you body and run this code, the after-effects should be:

Hi from parent's MethodC

Hi from sibling's MethodD

Hi from Child's MethodE

Hi from sibling's MethodC

Hi from sibling's changeless MethodS

Hi from Child's MethodV

Hi from GrandChild's MethodV

III. CONCLUSION

OK, so far so good. The wrapping cipher is a bit annoying but we can now finer accede accessible methods from an added ancestor chic (or as abounding such classes as we like). However, there are still some gaps in the accomplishing (for archetype inheriting adequate members).

References

1. R. Agrawal, L. DeMichiel, and B. Lindsay. Static type checking of multi-methods. In OOPSLA,pages 113–128, 1991.
2. E. Allen, D. Chase, J. Hallett, V. Luchangco, J. Maessen, S. Ryu, G. Steele, Jr., and S. Tobin-Hochstadt. The Fortress Language Specification, Version 1.0. Available at http://research.sun.com/projects/plrg/Publications/fortress.1.0.pdf, 2008. Accessed 3/09.
3. E. Allen, J. J. Hallett, V. Luchangco, S. Ryu, and G. L. Steele Jr. Modular multiple dispatch with multiple inheritance. In SAC ’07, pages 1117–1121. ACM, 2007.
4. D. Ancona, G. Lagorio, and E. Zucca. Jam - designing a Java extension with mixins. ACM Trans. Program. Lang. Syst., 25(5):641–712, 2003.
5. D. Ancona and E. Zucca. An algebraic approach to mixins and modularity. In Algebraic and Logic Programming, pages 179–193, 1996.
6. G. Baumgartner, M. Jansche, and K. Läufer. Half & Half: Multiple dispatch and retroactive abstraction for Java. Technical Report OSU-CISRC-5/01-TR08, Dept. of Computer and Information Science, The Ohio State University,March 2002.
7. S. Geetha, D. Jeya Mala – Object Oriented Analysis and Design Using UML, 1st Edition, McGraw-Hill Education.

AUTHOR(S) PROFILE

Manshi Eteshkumar Shah, received the B.E. degree in Computer Engineering from Veer Narmad South Gujarat University in 2011, respectively. She now with Parul University.

