Study on Function Ambiguity in Inheritance Using Object Oriented Programming with C++, Java and C#

Manoj R Chakravathi
Shilpa Medicare Limited, Raichur, India
manoj16mar93@gmail.com

Abstract:
C++ Java and C# is object oriented programming language [1] [2]. It supports features of object-oriented programming systems. Main features are Polymorphism, Encapsulation and Inheritance. Polymorphism is the attribute that allow one interface to control access to a general class of action. Encapsulation is a mechanism that binds both code and data together in a single unit like class such that to keep both safe from outside interfaces and misuse of code. Inheritance is a mechanism of creating new class by using old class. Old class are also called as Base Class, Super Class and Parent Class and New Class is also called as Derived Class, Subclass and Child Class. Inheritance allow the programmer to reuse the code. Once the programmer is prepared with his Class and tested, it can be used by other library programmers. While programming in this fashion problems like function ambiguity may arise, which are difficulty to identify and debug. This paper shows how to overcome with function ambiguity in inheritance in C++ [3] [4] [5], Java and C# [6].

Keywords: Class, Inheritance, Function Overriding, Function Overloading, Virtual Function, Virtual Base Class, Scope Resolution Operator

1. Introduction

During inheritance the major problem is function ambiguity it happens when multiple base classes are inherited in derived class, we have 5 different types of Inheritance. Namely Single Inheritance, Multiple Inheritance, Hierarchical Inheritance, Multilevel Inheritance, Hybrid Inheritance (also known as Virtual Inheritance). Function ambiguity will arise only when a more than one base class are inherited in derived class. In multiple Inheritance, there is a possibility that a class may inherit member functions with the same function name from two or more base classes and the derived class may not have same functions with same name as those of its in Base Classes. If the object of the derived class needs to access one of the member function with same one named of the base classes, then it results in ambiguity as it is not clear to the compiler to which bases class member function should be refer. The ambiguity simply means the state when the compiler is confused. The solution to this ambiguity can be done using two methods in C++ [7] [8] first is using Scope Resolution Operator second is by using Virtual Vase Function in C++. And in Java, C# solved by Interfaces Method.

2. Solution to function ambiguity in inheritance in C++

Consider a Base Class x with member function named display() and derived1 class and derived2 class are inherited from base class x and this two classes has a function with same function name, same arguments and with same return type as in the base class x. Another class y is derived from the class derived1 and class derived2. If suppose display () function is called with the help of class y object objectofy. display () then complier with pop up an error because when you inherit a classes base class members are copied in derived class, means here in our example display () is copied in two derived classes(derived1 and derived2)moreover explicitly we have mentioned display () function in derived1 and derived2 classes.

At the moment when object of class y access the member function display () as it is copied in two derived classes(derived1 and deried2)plus we have included explicitly since we have two display () in each derived class(that is derived1, derived2) since complier can't understand to which of the display () function should it invoke so it displays an error. When a function with same name, arguments, return types are present in same blocks they are called as function overloading and if they are present in two different blocks (like class) then they are called as Function Overridden. Fig .1 shows an Ambiguous Call Diagram and Fig.2 show an Ambiguous Call Program with respective to inheritance.

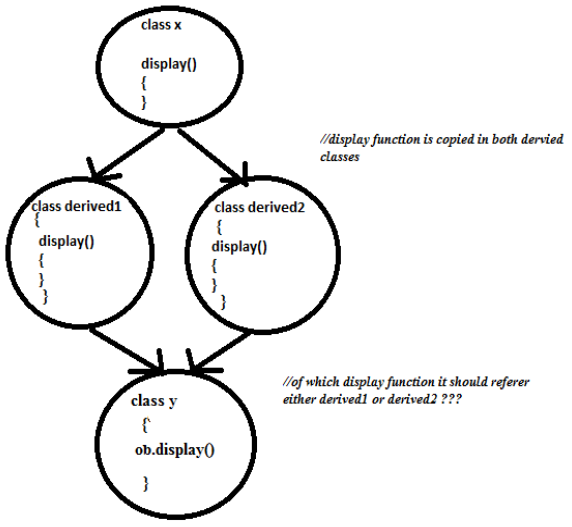

Fig. 1. Ambiguous Call Diagram

```
#include<iostream>
using namespace std;
class x
{
 public:
  void display()
    {
     cout<<"x class display()\n";
    }
 };
class derived1:public x
 {
  public:
   void display()
     {
      cout<<"derived1 class display()\n";
     }
  };
class derived2:public x
 {
  public:
   void display()
     {
      cout<<"derived2 class display()\n";
     }
 };
class y:public derived1, public derived2
 {};

int main()
{
 y object;
 object.display(); // this is a ambiguous..
}
```

Fig.2 Ambiguous Call Program Example in C++.

A. Using Scope Resolution Operator

The scope resolution operator(::) is a special operator in C++, is used in two different situation first is used to fetch global variables secondly is used to identify which member function belongs to which class.Fig.2 shows a ambiguous call program where a complier will Issus a error message when we try to compile it, this program can be made successfully run by using scope resolution operator, during invoking display() in main() use scope resolution operator like object name. class name::function name class name can be any class either dervied1, derived2 or class x which ever display() of class u wishes to invoke use that class name example if you wishes to use display() of derived2 than use like object.derived2::display() to invoke display() version of derived2.

```
#include<iostream>
using namespace std;
class x
{
 public:
  void display()
    {
     cout<<"x class display()\n";
    }
 };
class derived1:public x
 {
  public:
   void display()
     {
      cout<<"derived1 class display()\n";
     }
  };
class derived2:public x
 {
  public:
   void display()
     {
      cout<<"derived2 class display()\n";
     }
 };

class y:public derived1, public derived2
 {};

int main()
{
 y object;
 object.derived2::display(); // this is not ambiguous..
}
```

Fig. 3.

B. Virtual Base Class

Will consider another example to exam working of virtual base class which helps to overcome ambiguous function in inheritance in C++. Virtual is keyword in C++ and has two different version mainly is virtual function and another is virtual base class. Using Virtual Function, we can invoke different overridden function using same statements.

When we use virtual keyword attached to the first statements of class declaration than only one copy of base class members are copied in derived classes.

Hence virtual keyword is used to avoid duplicates copies. When two or more objects are derived from a common base class, we can use virtual keyword (virtual base class) to prevent multiple copies of the base class being present in an object of derived class by declaring parent class/ base class as virtual when it is used during inheritance. Such kind of parent/base class is commonly called as Virtual Base Class.

```
#include<iostream>
using namespace std;
class y
{
 public:
 int i;
};
class derived1 :  public y
{
 public:
 int j;
};
class derived2:  public y
{
 public:
 int k;
};
class derived3: public derived2, public derived1
{
 public:
 int sum;
};
int main()
{
 D ob;
 ob.i = 10; //ambiguous since multiple copies of i is inherited.
 ob.j = 20;
 ob.k = 30;
 ob.sum = ob.i + ob.j + ob.k;
 cout << "Value of i is : "<< ob.i<<"\n";
 cout << "Value of j is : "<< ob.j<<"\n"; cout << "Value of k is :"<< ob.k<<"\n";
 cout << "Sum is : "<< ob.sum <<"\n";
 return 0;
}
```

Fig. 4.

The above program show in Fig.4 will pop up an error because of i variable it has been copies in derived1 and derived2 classes during compilation when invoking ob. complier will get into the confusion of which i it should invoke whether of class y/ class derived1/class derived2 so complier will Pop up you an error.

The use of virtual base class is shown above in Fig 5, the program which is shown in Fig 5will give you an output without any error because of virtual keyword mentioned in declaration of class, Keyword virtual will make sure you that only one copy of base class contains will be copied in derived class during inheritance.

```
#include<iostream>
using namespace std;
class y
{
 public:
 int i;
};
class derived1 : virtual public y
{
 public:
 int j;
};
class derived2: virtual public y
{
 public:
 int k;
};
class derived3: public derived2, public derived1
{
 public:
 int sum;
};

int main()
{
 D ob;
 ob.i = 10; //unambiguous since only one copy of i is inherited.
 ob.j = 20;
 ob.k = 30;
 ob.sum = ob.i + ob.j + ob.k;
 cout << "Value of i is : "<< ob.i<<"\n";
 cout << "Value of j is : "<< ob.j<<"\n"; cout << "Value of k is :"<< ob.k<<"\n";
 cout << "Sum is : "<< ob.sum <<"\n";
 return 0;
}
```

Fig. 5.

3. Solution to Function Ambiguity in Inheritance in Java and C#

When speaking about java -a pure object-oriented programming [9] [10][11] [13] language, provides a lot solution to the drawback of C++ languages (example pointers, memory management) But the C# language provides pointers to programmers. The reasons for omitting multiple inheritance directly from the Java and C# language is mostly for doing Java and C# as the simple and easy object-oriented language. As a simple language, Java's creators wanted a language that most developers could grasp without extensive training.

So together, they worked to make the language as similar to C++ as possible (familiar) without carrying over C++ [12] unnecessary complexity (simple). In the point of designers' opinion, multiple inheritance causes more problems in C++ as we discussed in paper and confusion with solution. Classes are just permitted just to acquire from the single base class which is called single inheritance both in Java and C#.

So, they removed multiple inheritance with direct implementation from the language so make Java [13] [14] and C# [15] simple, Easy and Understandable to everyone. Java developer introduced language construct called interfaces to implement multiple inheritance. Which c# developers also followed same interfaces concept to implement multiple inheritance.

Java interface issimilar to class except interfaces will contain only static variable, method without body(definition) and objects cannot be created by using interface.A class implements any number of interface but one interfaces can extends another interface. There are different types interface inheritance that is Single, Multiple, Multilevel, Hybrid.Interface keyword isused to create interface and keyword implements is used to implement interface to class.

Interface Summary

- Interface functions or methods should be public and abstract.
- Interface Fields or Variables Should Be in Either Public or Final.
- Use Keyword Interface to Create an Interface
- Class Which Implementing an Interface Should Always Use Keyword Implements.
- Objects cannot be created from Interface.
- Interfaces Basically Don't Have Constructors.
- An Interface Can Extends One or More Than One Interfaces.

```
public interface Nmit
{
  public string Ise="hello";
  public void HelloNmit();
}
```

Fig. 6. Java syntax of interface example

```
public class MyNmit implements Nmit
{
   public void HelloNmit()
   {
     System.out.println("My hello");
    }
}
```

Fig. 7. shows java implementations interface to class

```
public interface Nmit
{
 public string Ise="hello";
 public void HelloNmit();
}
```

Fig. 8. c# example of interface syntax

```
public class MyNmit : Nmit
{
  public void HelloNmit()
  {
    console.writeline("hello")
   }
}
```

Fig. 9. c# implementation interface to class

```
interface Nmitcar
  {
    int  speed=90;
    public void distance();
  }
interface Nmitbus
  {
   int distance=100;
   public void speed();
  }
class Nmitvehicle implements Nmitcar,Nmitbus
  {
   public void distance()
    {
     int  distance=speed*100;
     System.out.println("distance travelled is"+distance);
    }
   public void speed()
    {
     int speed=distance/100;
    }
  }

class main
 {
   public static void main(String args[])
    {
     System.out.println("NmitVehicle");
     Nmitvechicle Ise=new Nmitehicle();
     Ise.distance();
     Ise.speed();
     }
 }
```

Fig. 10. example of multiple inheritance in java

```
interface Nmitcar
  {
    int  speed=90;
    public void distance();
  }
interface Nmitbus
  {
   int distance=100;
   public void speed();
  }
class Nmitvehicle : Nmitcar,Nmitbus
  {
   public void distance()
    {
     int  distance=speed*100;
     System.out.println("distance travelled is"+distance);
    }
   public void speed()
    {
     int speed=distance/100;
    }
  }

class main
 {
   public static void main(String args[])
    {
     System.out.println("NmitVehicle");
     Nmitvechicle Ise=new Nmitehicle();
     Ise.distance();
     Ise.speed();
     }
 }
```

Fig. 11. example of multiple inheritance program in c#

The difference between implementation language construct in these two languages is Java uses implement keyword to implement interface but in C# is uses: to implement interface to class.

4. Conclusion

This paper shows the concept of function ambiguity in inheritance with three different object-oriented programming language C+, Java and C# and also shows how to overcome by Diamond Inheritance Problem with code. In C++ by using Scope Resolution Operator (: :) and Virtual Base Class and in Java and C# by using Interfaces Concept.

REFERENCES

[1] A Study on Inheritance Using Object Oriented Programming with C++. Volume 1, Issue 2, July 2013 International Journal of Advance Research in Computer Science and Management Studies.

[2] C++ Report and Journal of Object-Oriented Programming (partial) archive
http://www.adtmag.com/joop/index.asp.

[3] Todd Veldhuizen: Techniques for Scientific C++
http://www.extreme.indiana.edu/~tveldhui/papers/techniques/techniques.html

[4] Quinn Tyler Jackson's papers
http://qtj.n3.net/~quinn/

[5] The new C++ casting operators
http://www.acm.org/crossroads/xrds3-1/ovp3-1.html

[6] Interviews with Nathan Myers and Stan Lippmann of state of C++
http://www.technetcast.com/tnc_cpp0.html

[7] Paul Pedriana: High Performance Game Programming in C++
http://www.ccnet.com/~paulp/index.html

[8] DENDROCLIM2002: AC++ program for statistical calibration of climate signals in tree-ring chronologies$ Franco Biondia,*, KishorWaikulb a Department of Geography, University of Nevada, Mail Stop 154, Reno, NV 89557-0048, USA bDepartment of Computer Science, University of Nevada, Reno, NV 89557, USA Received 11 March 2003; received in revised form 4 November 2003; accepted 4 November 2003

[9] Java, Java, Java-C.L. Sabharwal , Dept. of Computer Sci., Missouri Univ., Rolla, MO, USA IEEE Potentials (Volume: 17 , Issue: 3 , Aug/Sep 1998)

[10] Parallelism in object-oriented languages: a survey
-B.B. WyattComput. Sci. & Eng., Texas Univ., Arlington, TX, USA
K. KaviComput. Sci. & Eng., Texas Univ., Arlington, TX, USA
S. HufnagelComput. Sci. & Eng., Texas Univ., Arlington, TX, USA- IEEE Software (Volume: 9 , Issue: 6 , Nov. 1992)

[11] Implementing concurrent object-oriented languages on multicomputers - A. Yonezawa ; S. Matsuoka ; M. Yasugi ; K. Taura IEEE Parallel & Distributed Technology: Systems & Applications (Volume: 1 , Issue: 2 , May 1993)

[12] Avoiding Insecure C++ —How to Avoid Common C++ Security Vulnerabilities Aaron Ballman; David Svoboda 2016 IEEE Cybersecurity Development (SecDev)Year: 2016Page s: 65 – 65IEEE Conferences

[13] Introducing embedded systems in the first C/C++ programming class James O. Hamblen; Zachery C. Smith; Winne W. Woo 2013 IEEE International Conference on Microelectronic Systems Education

(MSE)Year: 2013Page s: 1 – 4Cited by: Papers (3)IEEE Conferences

[14] c# Blueprint Action Pattern Peng Gao; Jianbin Liu 2013 International Conference on Information Science and Cloud Computing Companion Year: 2013Page s: 376 – 381

[15] Portable C/C++ code for portable XML data Zhaoqing Wang; H.H. Cheng IEEE Software Year: 2006 , Volume: 23 , Issue: 1