ISSN:0975-9646

Diamond Effect in Object Oriented Programming Languages

Rajesh Jangade, Anil Barnwal, Satyakam Pugla

Amity Institute of Biotechnology,
Amity University, Noida, Uttar Pradesh, India

***Abstract*---Now a day the maximum percentage of the people from the world of computer programmers are using object oriented programming languages like C++, Java, Eiffel, Dot Net and etc. Why, because these languages are incorporated with some easiness, advanced, re-useful and needful features. And Inheritance is most important portion of any object oriented programming languages that plays vital role to like the language by programmers. Here I am going to discuss some basics problems and their solutions related to a situation of multiple inheritance called Diamond Effect. This problem comes in existence when we use multiple inheritance where one class is inherited by two different sub classes and again these two subclasses are inherited by any single class. Multiple inheritance in object oriented language like C++ is a powerful, but tricky tool, that often leads to problems if not handled carefully. Diamond effect occurs in most of the object oriented languages and each language has some different syntax for coding. In this paper whatever coding is used, will be in the reference of C++.**

***Keywords*: OOP, inheritance, diamond effect, base class, derived class and virtual base class.**

I. INTRODUCTION

Introduction of diamond effect includes so many things and topics that we have to understand first. Here I am trying to emphasize on the topics which will be coming in the way of diamond effect problems. Rest of all other topics will be discussed lightly sometimes [7][8]. The main path is

OOP Language => Inheritance => Multiple Inheritance => Diamond Effect

Introduction of Object Oriented Programming Language:

Any computer language which supports the following features is called object oriented programming language:

A. Class and Object

Class is a collection of properties of anything where property includes each and everything related to that thing. Whereas object is a real time entity which properties are described or written inside the class. In other word we can say that object is just an instance of the predefined class.

B. Data Abstraction

Data abstraction is a way to show only required properties of anything and hide the other details or properties which are not required currently. Example: Electric power supply system in a building, the people who are living in the building are given only switch boards to use. We know which switch is for fan and witch switch is for tube light and so on. If we need to on the fan then we just need to press the switch which is connected to fan only. Here we don't need to know about the internal wire connection the switch board. Here we are given switches to use and all others details of the switch board is already hidden, is called data abstraction.

C. Data Encapsulation

Data Encapsulation is way to bind or wrap the related things together. In together word we can say that all the properties of any things should wander together. Whenever we need any of the properties we just need to search the thing and then we call desired property, instead of searching the property and the things separately.

D. Inheritance

Inheritance is a way to use the properties of predefined class in the new classes. We can say, it provides a facility which is known as reusability in area of computer languages. For example we know any person is inherited by his son so his son can use all or some of properties of father. In such case, the class which properties are being used by is called base/parent/super class and the class which is using properties is called derived/child/sub class. The types of inheritance and their properties will be discussed later in paragraph 1.2.

II. CONCEPT OF INHERITANCE

It is already briefly discussed in paragraph 1.1.4; inheritance is a way to use the properties of predefined class in the new classes [1][3].

Inheritance is denoted by arrows as shown in Fig. 1 that shows connectivity between two classes.

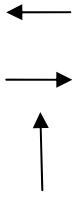


Fig. 1 Arrows: used to connect two classes.

In coding the special symbol colon (:) is used as symbol of inheritance.

There are four types of inheritance.

A. Simple Inheritance

When any single class inherits only one class then it is called simple or single inheritance.

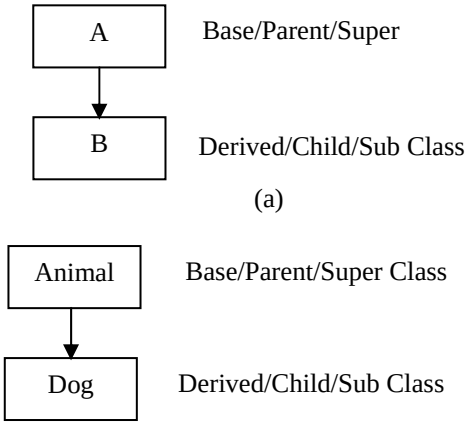


Fig.2 (a) and (b) show the two forms of simple inheritance.

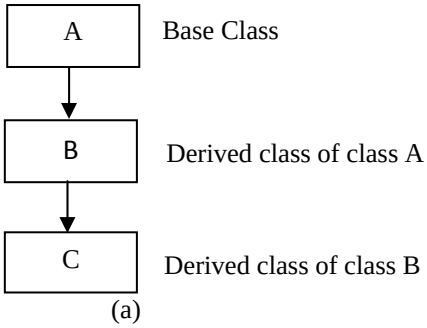
In Fig. no 2(a) one class A is inherited by other class B, similarly in Fig. 2(b) class Dog inherits the properties of class Animal.

Basic Syntax:

```
class A

{
    Some Member Data
    Some Member Functions
};

class B : public A
{
    Some Member Data
     Some Member Functions
};
class Animal

{
    Some Member Data
    Some Member Functions
};
class Dog : public Animal
{
    Some Member Data
    Some Member Functions
};
```

B. Multilevel Inheritance

When one class is inherited by second class and again this second class is inherited by third class then it is called multilevel inheritance. It is diagrammatically shown in Fig. 3 [1][3].



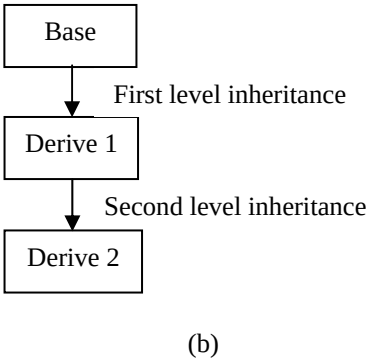


Fig. 3 (a) and (b) showing the example of multilevel inheritance

Basic Syntax:

```
class A

{
        Some Member Data
        Some Member Functions
};

class B :  public A
{
        Some Member Data
        Some Member Functions
};
class C :  public B
{
        Some Member Data
        Some Member Functions
};
```

C. *Hierarchical Inheritance*

When one class is inherited by more than one class then it is called hierarchical inheritance.

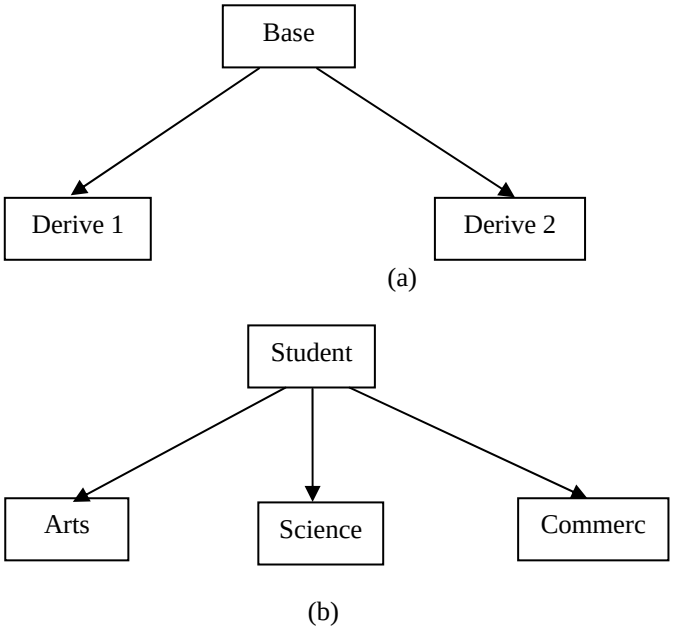


Fig. 4(a) and (b) showing the examples of multilevel inheritance

In Fig. 4(b) it is shown that students of arts, science and commerce are ultimately students so these tree different classes *Arts*, *Science* and *Commerce* inherit the properties of one base class *Student*.

Basic Syntax:

```
class Student

{
        Some Member Data
        Some Member Functions
};
class Arts :  public Student
{
        Some Member Data
        Some Member Functions
};
class Science :  public Student
{
        Some Member Data
        Some Member Functions
};
class Commerce  : public Student
{
        Some Member Data
        Some Member Functions
};
```

D. Multiple Inheritance

When two or more than two classes are inherited by one class then it is called multiple inheritance[1][3][5]. In other words we can say that in multiple inheritance any single class inherits more than one base classes as shown in Fig. 5.

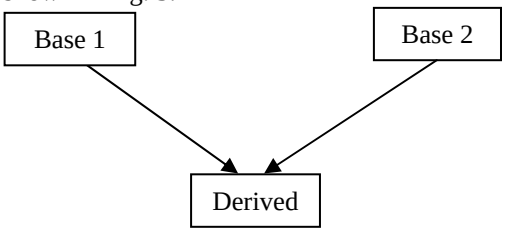


Fig. 5 Multiple Inheritance

Basic Syntax:

```
class Base1
{
        Some Member Data
        Some Member Functions
};
class Base2
{
        Some Member Data
        Some Member Functions
};
class Derived  : : public Base1, public
Base2
{
        Some Member Data
        Some Member Functions
};
```

The detail description of multiple inheritance is discussed in paragraph 1.3 with suitable examples

III. CONCEPT OF MULTIPLE INHERITANCE

We have above discussed that in multiple inheritance any single class uses the properties of more than one base class. In other words, there is a class being derived from many classes [1][3][5].

It is clear from Fig. 6(a) that result of the student is combined evaluation of both internal and external exams. So the *Result* class inherits the properties of both class *Internal Exam* and *External Exam*. Similarly in Fig. 6(b) there are two different classes *Animals* and *Pet*, and the third class *Cat* is a pet animal so this *Cat* class uses the properties of both *Animals* and *Pet classes.*

Basically the Diamond Effect is a situation occurs in the case of multiple inheritance and will be discussed later.

Basic Syntax:

```
class InternalExam

{
        Some Member Data
        Some Member Functions
};
class ExternalExam
{
        Some Member Data
        Some Member Functions
};
class Result   : public InternalExam,
public ExternalExam
{
        Some Member Data
        Some Member Functions
};
```

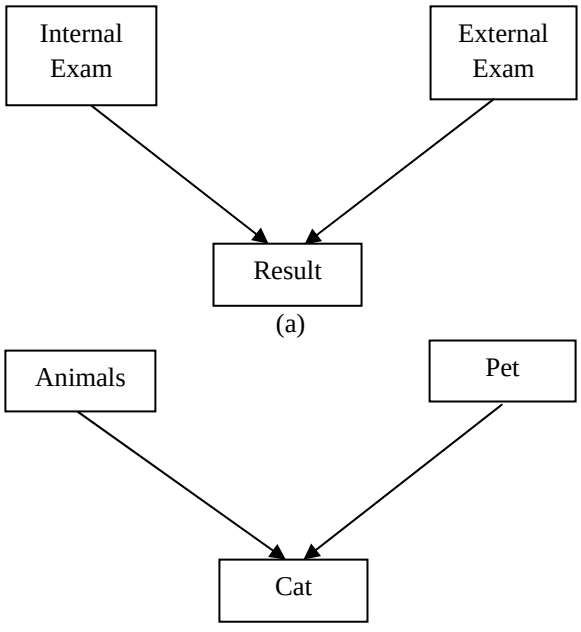


Fig. 6 Examples of multiple inheritance

Now let us see the copy of program of above example:

```
//Declaration of class InternalExam
class InternalExam
{
public:
char subject[5][20];
                    int Imark[5];
void getInternalDetails();
                    void showInternalDetails();
};

//Declaration of class ExternalExam
class ExternalExam
{
public:
char subject[5][20];
        int Emark[5];
void getExternalDetails();
                    void showExternalDetails();
};
//Declaration of class Result
class Result : public InternalExam, public ExternalExam
{
public:
char subject[5][20];
                    int Tot_Mark[5];
                    void getResult();
                    void showResult();
};

// Definition of class InternalExam
void InternalExam : : getInternalDetails()
{
        cout<<"\nEnter 5 name of subjects and marks obtained
by student\n";
                    for(int i=0;  i<5; i++)
                    {
                            cout<<"Enter        Subject Name:";
                            cin>>subject[i];
                            cout<<"Enter        Marks:";
                            cin>>Imark[i];
                    }
}
void InternalExam :  : showInternalDetails()
{
        cout<<"\nSubject \t\t Marks\n";
                    for(int i=0;  i<5; i++)
                    {
                            cout<<endl<<subject[i];
                            cout<<"\t\t"<<Imark[i];
                    }
}

// Definition of class ExternalExam
void ExternalExam : : getExternalDetails()
{
        cout<<"\nEnter 5 name of subjects and marks obtained
by student\n";
                    for(int i=0;  i<5; i++)
                    {
                            cout<<"Enter        Subject Name:";
                            cin>>subject[i];
                            cout<<"Enter        Marks:";
                            cin>>Emark[i];
                    }
}
void ExternalExam : :  showExternalDetails()
{
        cout<<"\nSubject \t\t Marks\n";
                    for(int i=0;  i<5; i++)
                    {
                            cout<<endl<<subject[i];
                            cout<<"\t\t"<<Emark[i];
                    }
}

// Definition of class Result
void Result : : getResult()
{
        cout<<"\nEnter 5 name of subjects \n";
                    for(int i=0;  i<5; i++)
                    {
                            cout<<"Enter        Subject Name:";
                            cin>>subject[i];
                    Tot_Mark[i] = Imark[i] + Emark[i];
                    }
}
void Result: :  showResult()
{
        cout<<"\nSubject \t\t Marks\n";
                    for(int i=0;  i<5; i++)
                    {
                            cout<<endl<<subject[i];
                            cout<<"\t\t"<<Tot_Mark[i];
                    }
}
```

In the above program in function *geResult()* it is shown that *Tot_Mark* are the sum of *Imark* and *Emark* of the corresponding *subject*. Here *Imark* and *Emark* are the members of class *InternalExam* and *ExternalExam* subsequently but being used in function *getResult()* of class *Result*. The *getResult()* is able to access these variables only because of the inheritance.

IV. INTRODUCTION OF DIAMOND EFFECT

Now let us discuss about a modified situation of above program where there are four classes:

- Student
- InternalExam
- ExternalExam
- Result

The *Student* includes some personal details, *internalExam* include details of internal exam, *ExternalExam* include details of external exam and class *Result* include the result of the students as shown above [4][6][7].

Here the class *Student* is inherited by both the classes *InternalExam* and *ExternalExam* using hierarchical inheritance and again these two classes *InternalExam* and *ExternalExam* are inherited by single class *Result* using multiple inheritance. And above both situations are shown in Fig. 7.

Here both the situations are shown in two separate Figures. Let's try to transform the Fig. 7 in form of single Fig. and in combined form of hierarchical and multiple inheritance and Fig. 7 becomes now Fig. 8. It is cleared from Fig. 8 that, it looks like a diamond and this way of representing the structure of classes is called ***diamond effect***.

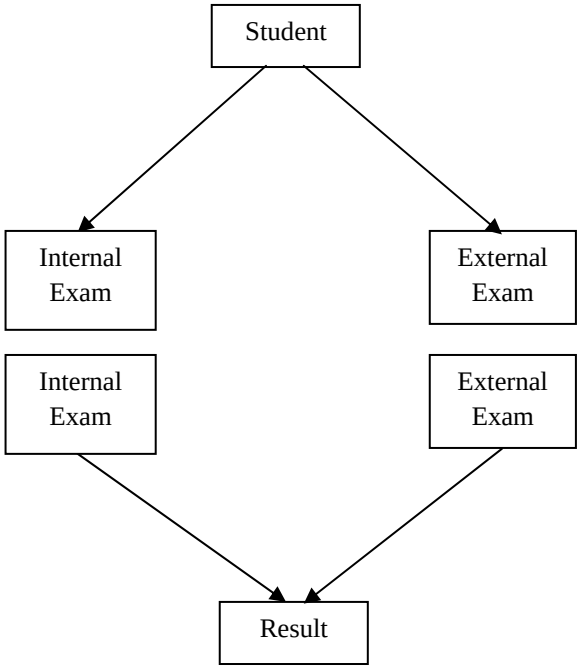


Fig. 7 Showing hierarchical and multiple inheritances separately

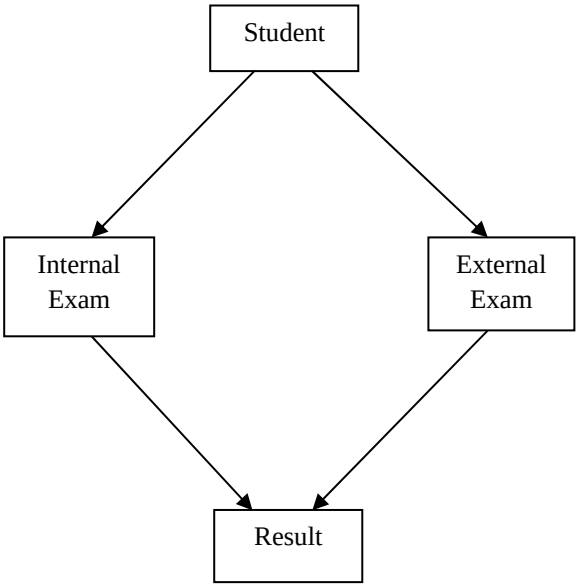


Fig. 8 Showing an aspect of Diamond Effect

In technical word we can define the diamond effect as ***"when one base class is having two different derived classes and again these derived classes have common and single derived class then this situation is called Diamond Effect in object oriented programming language."[3][7]*** In Fig. 8 one class S*tudent* is having two different derived classes *InternalExam and ExternalExam* and again these two derived classes is having single derived class *Result.* Classes *InternalExam* and *ExternalExam* are child class of base class *Student* and at same time these two classes are also base class of one common derived class *Result.*

Basic Syntax:

```
class Student
{
        Some Member Data
        Some Member Functions
};
class InternalExam : public Student


{
        Some Member Data
        Some Member Functions
};
class ExternalExam : public Student
{
        Some Member Data
        Some Member Functions
};
class Result   : public InternalExam,
public ExternalExam
{
        Some Member Data
        Some Member Functions
};
```

Copy of the program of classes *InternalExam, ExternalExam and Result* has already been written in paragraph 1.3. So here I am giving the copy of program of class Student only.

//Declaration of class *Student*

```
class Student
{
public:
char SName[20];
 int Roll_No;
          char course[20];
 void getStudentDetails();
          void showStudentDetails();
  };

//Definition of class Student
void Student : : getStudent()
{
          cout<<"\nEnter Name of Student\n";
          cin>>SName;
          cout<<"\nEnter Roll Number of Student\n";
          cin>>Roll_No;
          cout<<"\nEnter Course of Student\n";
          cin>>course;
};
void Student : : showStudent()
{
          cout<<"\nName of the Student:\t"<<SName;
          cout<<"\nRoll Number of the Student:\t"<<Roll_No;
          cout<<"\nCourse of the Student:\t"<<course;      };
```

V. ADVANTAGE OF DIAMOND EFFECT

In case of diamond effect, we say that there are at least four classes where one class has two child classes and these two children classes are having one common child class. As we know, in inheritance any derived class can use members of the base class [7][8].

Here the function *getInternalDetail()* can access the *getStudent()* and so on. Similarly the function *getExternalDetail()* can also access the *getStudent()* and so on. Let us see the modified version of *getInternalDetail()* and *getExternalDetail()* functions:

```
void InternalExam : : getInternalDetails()
{
        getStudent();          //accessing the function of base class
cout<<"\nEnter 5 name of subjects and marks obtained by student\n";
                for(int i=0;  i<5; i++)
                {
                        cout<<"Enter      Subject Name:";
                        cin>>subject[i];
                        cout<<"Enter      Marks:";
                        cin>>Imark[i];
                }
}
void ExternalExam : : getExternalDetails()
{
        getStudent();          //accessing the function of base class
        cout<<"\nEnter 5 name of subjects and marks obtained by student\n";
                for(int i=0;  i<5; i++)
                {
                        cout<<"Enter      Subject Name:";
                        cin>>subject[i];
                        cout<<"Enter      Marks:";
                        cin>>Emark[i];
                }
}
```

In the above program, ***highlighted*** lines show that *getInternalDetail()* and *getExternalDetail()* functions using the same function *getStudent().* It has no problem because both the classes *InternalExam* and *ExternalExam* have the same parent class *Student().*

Similarly the member functions of class *Result* can also access to the members of class *InternalExam* and *ExternalExam* because *Result* is child class of both the classes. Here are the modified versions of functions *getResult*() and *showResult().*

```
void Result : : getResult()
{
getInternalDetails();  // accessing the member function of class          // InternalExam
getExternalDetails(); // accessing the member function of class          //ExternalExam
cout<<"\nEnter 5 name of subjects \n";
                for(int i=0;  i<5; i++)
                {
                        cout<<"Enter      Subject Name:";
                        cin>>subject[i];
                Tot_Mark[i] = Imark[i] + Emark[i];
                }
}
```

```
void Result: :  showResult()
{
showInternalDetails();  // accessing the member function of class          // InternalExam
showExternalDetails(); // accessing the member function of class          //ExternalExam
        cout<<"\nSubject \t\t Marks\n";
                for(int i=0;  i<5; i++)
                {
                        cout<<endl<<subject[i];
                        cout<<"\t\t"<<Tot_Mark[i];
                }
}
```

There are again some ***highlighted*** lines which are showing that functions *getResult()* and *showResult()* are accessing the functions *getInternalExam(), getExternalExam()* and *showInternalExam(), showExternalExam(),* because the *Result* class is derived from both the base classes.

VI. PROBLEM ASSOCIATED WITH DIAMOND EFFECT

Before starting discussion about the problems associated with diamond effect concept let us try to restructure the Fig. 8 as shown in Fig. 9.

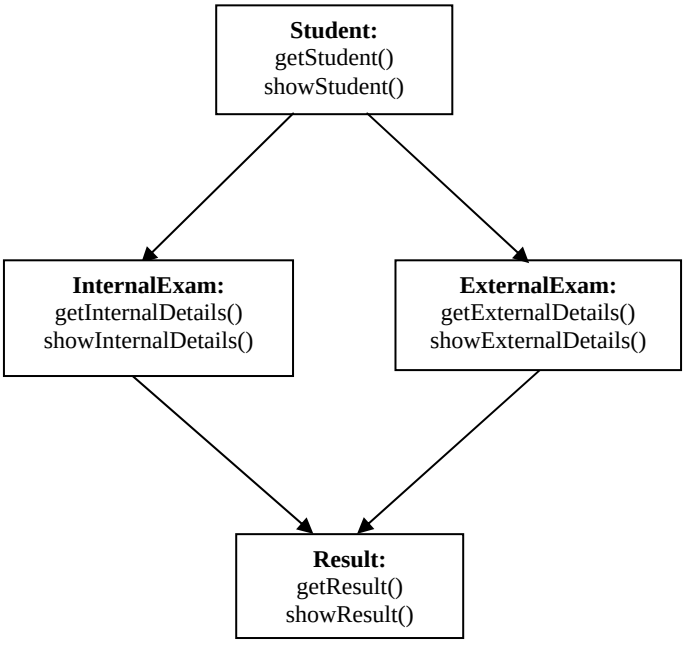


Fig. 9 Diamond Effect with class and their member functions.

As it is shown in Fig. 9, we can say that the class *Student* is grandparent of *Result* class hence the *Result* class can directly access the members of *Student* class as:

```
void Result : : getResult()
{
//getInternalDetails();
//getExternalDetails();
getStudent(); //Direct access to member of grandparent class(ERROR)
```

```
cout<<"\nEnter 5 name of subjects \n";
                for(int i=0;  i<5; i++)
                {
                        cout<<"Enter      Subject
Name:";
                        cin>>subject[i];
                Tot_Mark[i] = Imark[i] + Emark[i];
                }
}
void Result: :  showResult()
{
//showInternalDetails();
//showExternalDetails();
showStudent();//Direct access to member of grandparent
class(ERROR)
        cout<<"\nSubject \t\t Marks\n";
                for(int i=0;  i<5; i++)
                {
                        cout<<endl<<subject[i];
                        cout<<"\t\t"<<Tot_Mark[i];
                }
}
```

Error: there are two ways for *Result* class to access the members of *Student* class as:

Result->InternalExam->Student

and

Result->ExternalExam->Student

Then the question is by which way the members of *Student* class will be accessed? This situation is called **ambiguity problem** in the world of object oriented programming.

VII. SOLUTION OF PROBLEM ASSOCIATED WITH DIAMOND EFFECT

Here the virtual base class comes in the life. We make the class *Student* as virtual base class for the class *Result.* For this we use ***virtual*** keyword when create derived class [2][6][7]. So the following syntax is used to make virtual base class as:

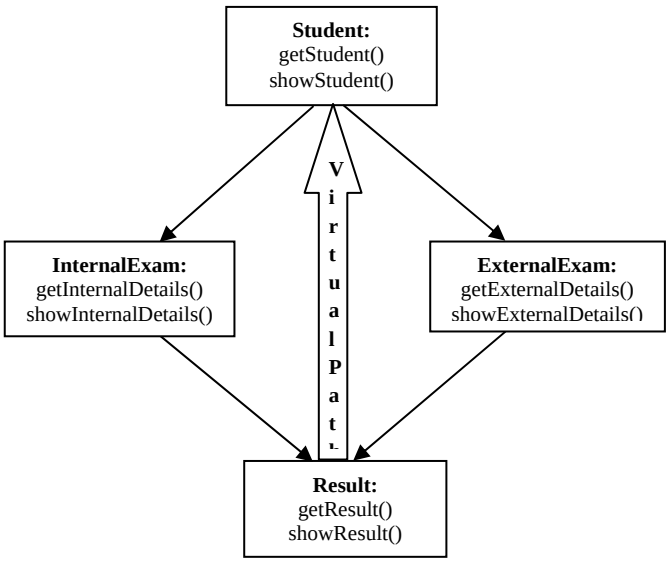


Fig. 10 Virtual Base Class

```
class Student
{
Some Member Data
        Some Member Functions
};
class InternalExam : virtual public Student

{
Some Member Data
        Some Member Functions
};
class ExternalExam : virtual public Student
{
 Some Member Data
 Some Member Functions
};
class Result   : public InternalExam, public ExternalExam
{
 Some Member Data
 Some Member Functions
          };
```

By using above syntax the classes *InternalExam* and *ExternalExam* will create a virtual path for the next derived class *Result* to first base class *Student*. Fig. 10 shows diagrammatic form of virtual base class.

And the above given program with error syntax is rectified as :

```
void Result : : getResult()
{
//getInternalDetails();
//getExternalDetails();
getStudent(); //Direct access to member of grandparent
class(No Error)

cout<<"\nEnter 5 name of subjects \n";
                for(int i=0;  i<5; i++)
                {
                        cout<<"Enter      Subject
Name:";
                        cin>>subject[i];
                Tot_Mark[i] = Imark[i] + Emark[i];
                }
}
void Result: :  showResult()
{
//showInternalDetails();
//showExternalDetails();
showStudent();//Direct access to member of grandparent
class(No Error)
        cout<<"\nSubject \t\t Marks\n";
                for(int i=0;  i<5; i++)
                {
                        cout<<endl<<subject[i];
                        cout<<"\t\t"<<Tot_Mark[i];
                }
}
```

So the concept of virtual base class is used to solve the problem generated in situation of diamond effect.

REFERENCES

[1] Donna Malayeri, Jonathan Aldrich, *CZ: Multiple Inheritance without Diamond Effect, in OOPSlA09*.

[2] What is a virtual base class?, *CareerRide.com*.

[3] G. Singh. *Single Versus Multiple Inheritance in Object Oriented Programming. SIGPLAN OOPS.*, (5): 34-43, 1994.

[4] A. Shalit. *The Dylan Reference Manual: The definitive Guide of the new object-oriented dynamic Language*. Addison Wesley, 1997.

[5] Mathew Cochran. *Coding Better: Using Classes V Interfaces*, January 18, 2009.

[6] V. Krishnapriya, Dr K Ramar. *Exploring the difference between object oriented class inheritance and Interface using coupling Measures*., 2010, International Conference on Advances in Computer Engineering.

[7] E. Balaguruswami. *Object Oriented Programming with C++*, 6th Edition, Tata McGraw-Hill Education.

[8] Herbert Schildt , *C++: The Complete Reference*, 4th Edition, 2002.