EPH - International Journal of Science And Engineering
ISSN (Online): 2454 2016
Volume 01 Issue 03 July 2015

DOI: https://doi.org/10.53555/eijse.v1i3.59

A STUDY ON OBJECT ORIENTED PROGRAMMING WITH C++

Navpreet singh 17857[1]*
**[1]Dronacharya college of engineering, India*

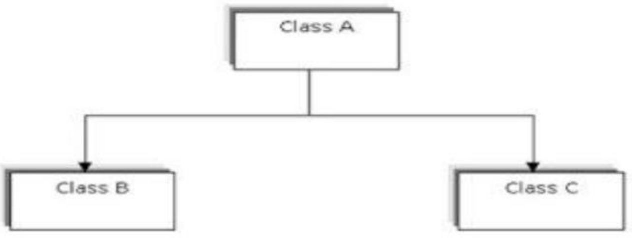
Abstract:-
C++ strongly supports the concept of Reusability. The C++ classes can be reused in severalways. Once a class has been written and tested, it can be adapted by another programmer to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones. The mechanism of deriving a new class from an old one is called INHERITANCE.

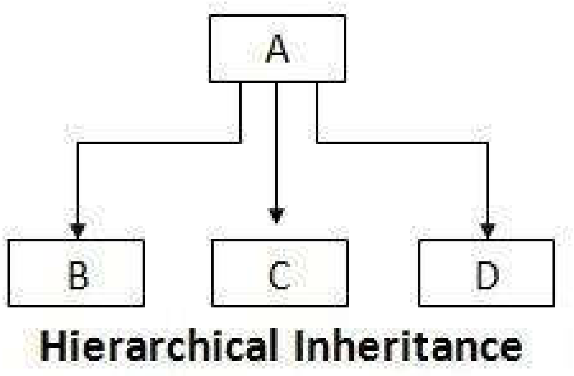
Keywords: - *Reusability, Base class –Subclass, Private data member, Public data member and Types of Inheritance.*

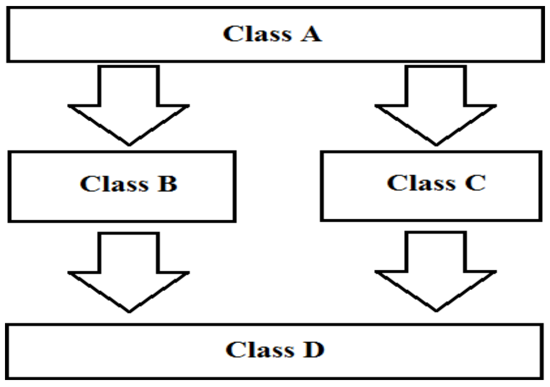
INTRODUCTION

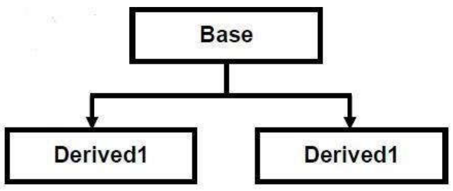
Inheritance (OOP) is when an object or class is based on another object (prototypal inheritance) or class (class- based inheritance), using the same implementation (inheriting from an object or class) specifying implementation to maintain the same behavior (realizing aninterface; inheriting behavior). It is a mechanism for code reuse and to allow independent extensions of the original software via public classes and interfaces. The relationships of objects or classes through inheritance giverise to a hierarchy. Inheritance should not be confused with subtyping.[2][3] Insome languages inheritance and subtyping agree,[a] whilein others they differ; in general subtyping establishes an is-a relationship, while inheritance only reuses implementation and establishes a syntactic relationship, not necessarily a semantic relationship (inheritance does not ensure behavioral subtyping). To distinguish these concepts, subtyping is also known as interface inheritance, while inheritance as defined here is known asimplementation inheritance or code inheritance.[4] Still, inheritance is a commonly used mechanism for establishing subtype relationships.[5]

Inheritance is contrasted with object composition, where one object contains another object (or objects of one class contain objects of another class); see composition over inheritance. Composition implements a has-a relationship, in contrast to the is-a relationship of subtyping.









IN THIS PAPER WE HAVE CONSIDERED FOLLOWING TYPES OFINHERITANCE:
1- SINGLE LEVEL INHERITANCE
2- MULTILEVEL INHERITANCE
3- HIERARCHIAL INHERITANCE
4- HYBRID INHERITANCE

1-SINGLE LEVEL INHERITANCE
When a single derived class is created from a single base class then theinheritance is called as single inheritance.

```
#include <iostream.h>
Class B
{int a;
Public:
int b;
void get_ab();
int get_a();
void show_a();
};
Class D:
Public B
{
int c; public:
void mul();
void display();
};
Void B :: get_ab()
{ a=5;b=10; }

Int B:: get_a()
{Return a;}
Void B:: show_a()
{Count<< "a="<<a<< "\n" ;}
Void D:: mul()
{c=b*get_a ();} Void D:: display ()
{
Count<< "a="<<get_a ()Count<< "b="<<b Count<< "c="<<c
}
int main ()
{
Dd;
d.get_ab();
d.mul ();
d.show_a();
display ();
d.b=20;
d.mul();
Display ();
Return 0;
```

2. HIERARCHICAL INHERITANCE
When more than one derived class are created from a single baseclass, then that inheritance is called as hierarchical inheritance.

```
Class first
{
Int x=10, y=20;
Void display ()
{
System.out.println ("This is the method in class one");
System.out.println ("Value of X= "+x);
System.out.println ("Value of Y= "+y);
   }
}
Class two extends first
{
```

```
void add ()
{
System.out.println ("This is the method in class two");
System.out.println ("X+Y= "+(x+y));
}
}
Class three extends first
{
void mul ()
{
System.out.println ("This is the method in class three");
System.out.println ("X*Y= "+(x*y));                           }     }
Class Hier
{
Public static void main (String args [])
{
two t1=new two();
Three t2=new three (); t1.display (); t1.add (); t2.mul ();
}
}
```

3. MULTI LEVEL INHERITANCE

When a derived class is created from another derived class, then thatinheritance is called as multi-level inheritance.

```
Class A
{
A ()
{
System.out.println ("Constructor of Class A has been called");
}
}
Class B extends A
{B()
{
Super ();
System.out.println ("Constructor of Class B has been called");
}
}
Class C extends B
{
C ()
{
Super ();
System.out.println ("Constructor of Class C has been called");
}
}
Class Constructor Call
{
Public static void main (String [] args)
{
System.out.println ("------Welcome to Constructor call Demo------------------------------------------------------- ");
C objc = new C ();
}

}
```

4. HYBRID INHERITANCE

```
Class stud
{
Protected:
int rno;
Public:
Void getno (int n)
{
Rno=n;
}
```

```
Void display_rno ()
{
Cout<<"Roll_no="<<rno<<"\n";
}
};
Class test: Public stud
{
Protected: Int sub1, sub2;
Public:
Void get_mark (int m1, int m2)
{
Sub1=m1;  Sub2=m2;
}
Void display mark ()
{
Cout<<"sub1"<<sub1<<"\n"; Cout<<"sub2"<<sub2<<"\n";
}
};
Class sports
{
Protected:
Float score;
Public:
Void get_score (float s)
{
Score=s;
}
Void put_score ()
{
Cout<<"Sort :"<< score<<"\n";
}
};
Class result: public test, public sports
{

Float total;  Public:         Void display ();
};
Void result::display ()
{
Total=sub1+sub2+score;
display_rno ();
Display mark ();
put_score ();
cout<<" total score :"<< total<<"\n";
}
Int main ()
{
Result s r1 r1. Getno (123);     r1. get_mark (60, 80) r1.get_ score (6) r1.display ();
```

Any combination of single, hierarchical and multi-level inheritances iscalled as hybrid inheritance.

CONCLUSION-

Inheritance is one of the most important components of the object oriented programming and in turns the hierarchies of inheritance classes. By now the concept must have been cleared that inheritance isused with the base or parent class via the derived class. We have alreadyseen that c++ supports both simple and multiple inheritance. We havealso seen the three different types of inheritance. If you want the protected members in the base class to be the same as protected and public members to be the same as public in the derived class, or in otherwords to keep these two access specifiers with the derived class. But mostly the programmer is concerned with public inheritance. In general,this would be advisable never to leave out the public access specifier. One of the biggest advantages of inheritance is that with any access control specifier, a derived class can define its own members and also can redefine members of the base class. This means the derived class can override the inherited characteristics and can add new characteristcs.

At the end of the inheritance conclusion, inheritance is just a convenience for the programmer. If you really need simple inheritance,try to avoid complicated classes to overcome or minimize such situations. If you cannot avoid and really want to include multiple inheritance, you better avoid the more complicated situations. If you avoid all these and try to model the situation as simple as possible, butthere is no simpler.

References-

[1]. www.wikipedia.org
[2]. fcpc book
[3]. www.tutorialspoint.com
[4]. www.cppforschool.com