ISSN: 2321-7782 (Online)
Volume 3, Issue 1, January 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Concept of Reusability in C++ using Inheritance

Piyush Sudhakarrao Gondchawar
Computer Science & Engineering.
Prof. Ram Meghe Institute of Technology & Research,
Badnera, Amravati, India.

Abstract: C++ strongly supports the concept of Reusability. The C++ classes can be reused in several ways. 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.

The old class is referred to as the base class and the new one is called the derived class or subclass. A derived class includes all features of the generic base class and then adds qualities specific to the derived class.

In this paper, I have studied the Inheritance concept and its types using C++ (oop).

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

I. INTRODUCTION

Inheritance is the process by which objects of one class acquire the properties of objects of another class in the hierarchy.

Subclasses provide specialized behavior from the basis of common elements provided by the super class. Through the use of inheritance, programmers can reuse the code in the super class many times.

Once a super class is written and debugged, it need not be touched again but at the same time can be adapted to work in different situations. Reusing existing code saves time and money and increases a program's reliability.

For example, the scooter is a type of the class two-wheelers, which is again a type of (or kind of) the class motor vehicles. As shown in the below diagram the principle behind it is that the derived class shares common characteristics with the class from which it is derived.

New classes can be built from the existing classes. It means that we can add additional features to an existing class without modifying it. The new class is referred as derived class or subclass and the original class is known as base classes or super class.

Advantages of Inheritance

» Reusability

» Extensibility

In this paper we have considered the following types of Inheritance:

1. Single Level Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance

5. Hybrid Inheritance

1. Single Level Inheritance :

A derived class with only one base class is called single inheritance. Consider a simple example of single inheritance. In this program show a base class B and derived class D. The class B contains one private data member, one public data member, and three public member functions. The class D contains one private data members and two public member functions.

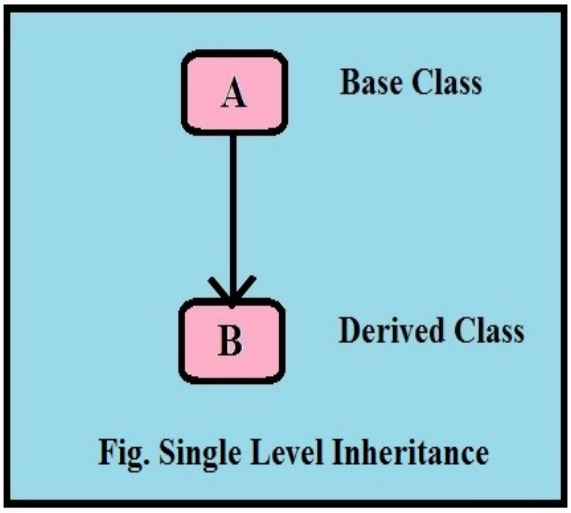

Fig. Single Level Inheritance

Example of Single Level Inheritance :

```
#include <iostream.h>
 class Value
{
protected:
int val;
public:
void set_values (int a)
{
 val=a;
}
};
 class Cube: public Value
{
public:
int cube()
{
 return (val*val*val);
}
```

```
};
 int main ()
{
Cube cub;
cub.set_values (5);
cout << "The Cube of 5 is::" << cub.cube() << endl;
return 0;
}
```

2. ***Multiple Inheritance :***

A class can inherit properties from more than one class which is known as multiple inheritances.

This form of inheritance can have several super classes. A class can inherit the attributes of two or more classes as shown below diagram.

Multiple inheritances allow us to combine the features of several existing classes as a starting point for defining new classes. It is like a child inheriting the physical features of one parent and the intelligent if another.

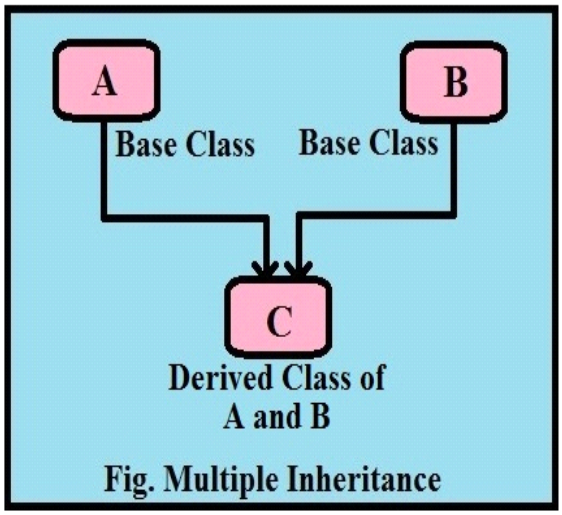

Fig. Multiple Inheritance

Example of Multiple Inheritances:

```
#include <iostream.h> using namespace std;
 class Square
{
protected:
int l;
public:
void set_values (int x)
{
 l=x;
```



```
}
};
 class CShow
{
public:
void show(int i);
};
void CShow::show (int i)
{
cout << "The area of the square is::" << i << endl;
}
class Area: public Square, public CShow
{
public:
int area()
{
 return (l *l);
}
};
 int main ()
{
Area r;
r.set_values (5);
r.show(r.area());
return 0;
 }
```

3. Hierarchical Inheritance :

When the properties of one class are inherited by more than one class, it is called hierarchical inheritance.

This form has one super class and many Subclasses. More than one class inherits the traits of one class. Additional members are added through inheritance is to use it as a support to the hierarchical design of a program. Many programming problems can be cast into a hierarchy where certain features of one level are shared by many others below that level.

In C++, problems can be easily converted into class hierarchies. The base class will include all the features that are common to the subclasses. A subclass can be constructed by inheriting the properties of the base class. A subclass can serve as a base class for the lower level classes and so on.

For example: bank accounts.

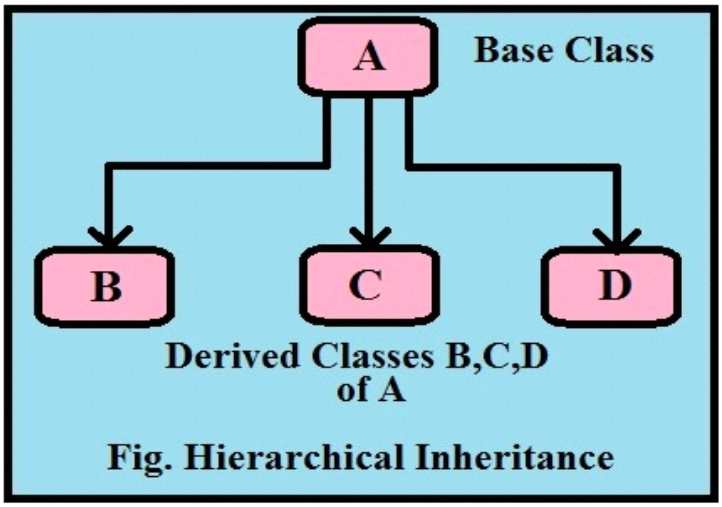

Fig. Hierarchical Inheritance

Example of Hierarchical Inheritance:

```
#include <iostream.h>
 class Side
{
protected:
int l;
public:
void set_values (int x)
{
 l=x;
}
};
 class Square: public Side
{
public:
int sq()
{
 return (l *l);
}
};
```

```
class Cube:public Side
{
public:
int cub()
{
 return (l *l*l);
}
};
 int main ()
{
Square s;
s.set_values (10);
cout << "The square value is::" << s.sq() << endl;
Cube c;
c.set_values (20);
cout << "The cube value is::" << c.cub() << endl;
return 0;
}
```

4. Multilevel Inheritance :

A class can be derived from another derived class which is known as multilevel inheritance.

Order of Constructor Calling in Multilevel Inheritance, when the object of a subclass is created the constructor of the subclass is called which in turn calls constructor of its immediate super class.

For example, if we take a case of multilevel inheritance, where class B inherits from class A, and class C inherits from class B, which show the order of constructor calling.

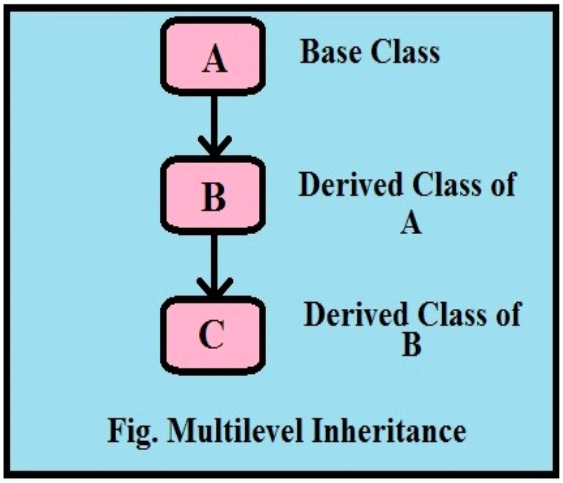

Fig. Multilevel Inheritance

Example of Multilevel Inheritance :

```
#include <iostream.h>
 class mm
{
protected:
int rollno;
public:
void get_num(int a)
{
 rollno = a;
}
void put_num()
{
 cout << "Roll Number Is:\n"<< rollno << "\n";
}
};
 class marks : public mm
{
protected:
int sub1;
int sub2;
public:
void get_marks(int x,int y)
{
sub1 = x;
sub2 = y;
}
void put_marks(void)
{
cout << "Subject 1:" << sub1 << "\n";
cout << "Subject 2:" << sub2 << "\n";
}
```



```
};
 class res : public marks
{
protected:
float tot;
public:
void disp(void)
{
tot = sub1+sub2;
put_num();
put_marks();
cout << "Total:"<< tot;
}
};
 int main()
{
res std1;
std1.get_num(5);
std1.get_marks(10,20);
std1.disp();
return 0;
}
```

5. Hybrid Inheritance :

There could be situations where we need to apply two or more types of inheritance to design one inheritance called hybrid inheritance.

For instance, consider the case of processing the student results, the weight age for sport is stored in separate classes called sports.

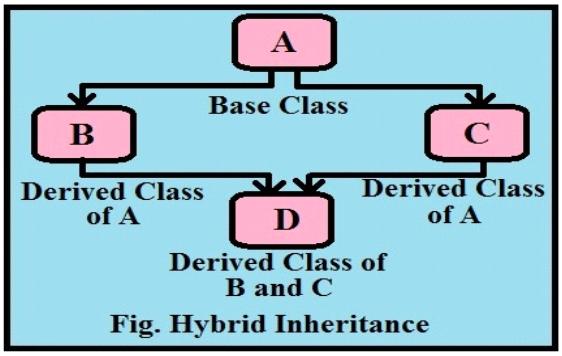

Fig. Hybrid Inheritance

Example of Hybrid Inheritance :

```
#include <iostream.h> class mm
{
protected:
int rollno;
public:
void get_num(int a)
{
 rollno = a;
}
void put_num()
{
 cout << "Roll Number Is:"<< rollno << "\n";
}
};
 class marks : public mm
{
protected:
int sub1;
int sub2;
public:
void get_marks(int x,int y)
{
sub1 = x;
sub2 = y;
}
void put_marks(void)
{
cout << "Subject 1:" << sub1 << "\n";
cout << "Subject 2:" << sub2 << "\n";
}
};
```



```
class extra
{
protected:
float e;
public:
void get_extra(float s)
{
e=s;
}
void put_extra(void)
{
 cout << "Extra Score::" << e << "\n";
}
};
class res : public marks, public extra
{
protected:
float tot;
public:
void disp(void)
{
tot = sub1+sub2+e;
put_num();
put_marks();
put_extra();
cout << "Total:"<< tot;
}
};
int main()
{
res std1;
std1.get_num(10);
```

```
std1.get_marks(10,20);

std1.get_extra(33.12);

std1.disp();

return 0;

}
```

II. CONCLUSION

The mechanism of deriving a new class from an old class is called inheritance; it provides the concept of Reusability that is the most important concept in C++. All types of inheritance with its own features and its use to provide users to reusability concepts strongly, to give save time and reduce the complexity.

Here, in this paper we have to study the above five types of inheritance. We have to find that inheritance is central concepts in C++ that allows deriving a class from multiple classes at a time.

References

1. E Balagurusamy, Object oriented Programming with C++, 6th Edition, New Delhi: Tata McGraw-Hill Publishing Company Limited.
2. Nell B. Dale, Studyguide for C++ Plus Data Structure, Academic Internet Publishers.
3. Yashavant Kanetkar, Test your C++ Skills, 1st Edition, BPB Publication.
4. Salivahanan S, Arivazhagan S – Digital Circuits and Design 4th Edition, Vikas Publishing House Pvt Ltd.
5. Yashwant Kanetkar, Let Us C++ Solutions – 2010 , BPB Publication.
6. S. Geetha, D. Jeya Mala – Object Oriented Analysis and Design Using UML, 1st Edition, McGraw-Hill Education.
7. Yashwant Kanetkar, Data Structures Through C++ 1st Edition, BPB Publication.
8. ISRD Group, Data Structures Through C++ 1st Edition, McGraw-Hill Education.
9. Bjarne Stroustrup, The C++ Programming Language, Person Publication.
10. Kyle Loudon, C++ Pocket Reference 1st Edition, O'reilly Publication.
11. Yashavant Kanetkar, C++ Programming, 1st Edition, BPB Publication.

