Inheritance in C++

Professor Usha Batra

Chairperson CS & IT

Dean, Corporate Relations & Engagement

Inheritance

- Reusability is yet another feature of OOP. C++ strongly supports the concept of reusability.
- The C++ classes can be used again in several ways. Once a class has been written and tested, it can be adopted by another programmers.
- This is basically created by defining the new classes, reusing the properties of existing ones.
- The mechanism of deriving a new class from an old one is called 'INHERTTENCE’.
- The old class is called 'BASE' class and the new one is called 'DERIEVED'class.

Derived Classes

The class that inherits or derives attributes and behaviour from another class is known as derived class. A derived class is specified by defining its relationship with the base class in addition to its own details.

The general syntax of defining a derived class is as follows:

class d_classname : Access specifier baseclass name

{ __

__ // members of derived class };

Visibility mode describes the status of derived features e.g.

class xyz //base class

{ members of xyz };

class ABC : public xyz //public derivation

{ members of ABC };

class ABC : XYZ //private derivation (by default)

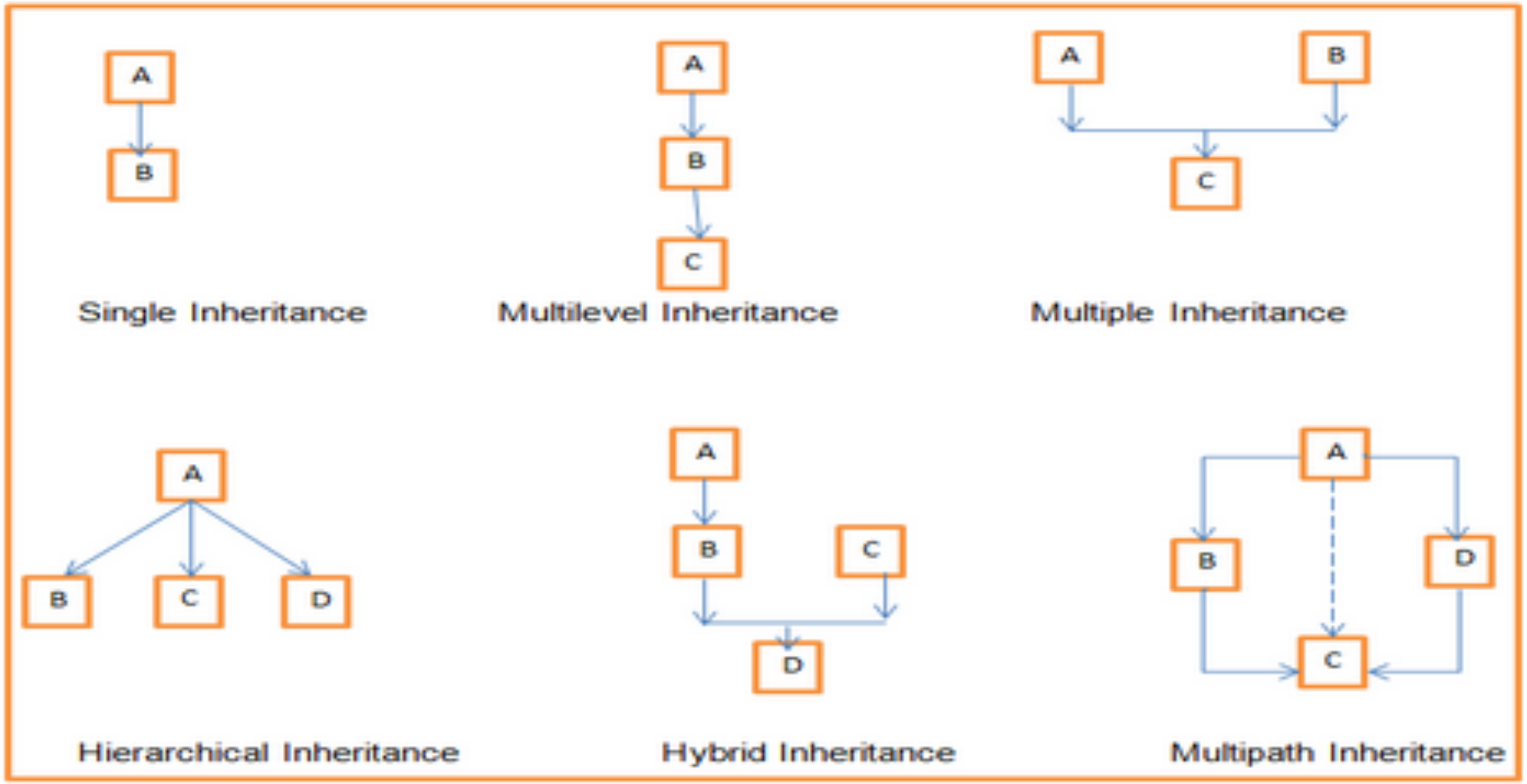
{ members of ABC };

Types of Inheritance

In C++ Inheritance is classified into following forms:

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance
- Multiple Inheritance
- Multipath Inheritance

Types of Inheritance



Single Inheritance: Public Derivation

```
#include<iostream.h>
#include<conio.h>
class worker
{
int age;
char name [10];
public:
void get ( );
};
void worker : : get ( )
{
cout <<"yout name please"
cin >> name;
cout <<"your age please" ;
cin >> age;
}
void worker :: show ( )
{
cout <<"In My name is :"<<name<<"In My age is :"<<age;
}
class manager :: public worker //derived class (publicly)
{
int now;
public:
void get ( ) ;
void show ( ) ;
};
void manager : : get ( )
{
worker : : get ( ) ; //the calling of base class input fn.
cout << "number of workers under you";
cin >> now;
cin>>name>>age;
} ( if they were public ) void manager :: show ( )
{
worker :: show ( ); //calling of base class o/p fn.
cout <<"in No. of workers under me are: " << now; }
main ( )
{
clrscr ( ) ;
worker W1;
manager M1;
M1 .get ( );
M1.show ( ) ;
}
```

Single Inheritance: Private Derivation

```
#include<iostream.h>
#include<conio.h>
class worker //Base class declaration
{
int age;
char name [10] ;
public:
void get ( ) ;
void show ( ) ;
};
void worker : : get ( )
{
cout << “your name please” ;
cin >> name;
cout << “your age please”;
cin >>age;
}
void worker : show ( )
{
cout << “in my name is: “ <<name<< “in” << “my age is : “ <<age;
}
class manager : worker //Derived class (privately by default)
{
int now;
public:
void get ( ) ;
void show ( ) ;
};
void manager : : get ( )
{
worker : : get ( ); //calling the get function of base
cout << “number of worker under you”; class which is
cin >> now;
} void manager : : show ( )
{
worker : : show ( ) ;
cout << “in no. of worker under me are : “ <<now;
} main ( )
{
clrscr ( ) ; worker wl ; manager ml; ml.get ( ) ; ml.show ( );
}
}
```

Single Inheritance: Protected Derivation

```
#include<conio.h>
#include<iostream.h>
class worker //Base class declaration
{ protected:
int age; char name [20];
public:
void get ( );
void show ( );
};
void worker :: get ( )
{
cout >> “your name please”;
cin >> name;
cout << “your age please”;
cin >> age;
}
void worker :: show ( )
{
cout << “in my name is: “ << name << “in my age is “ <<age;
}
class manager:: protected worker // protected inheritance
{
int now;
public:
void get ( );
void show ( ) ;
};
void manager : : get ( )
{
cout << “please enter the name In”;
cin >> name;
cout<< “please enter the age In”;
//Directly inputting the data
cin >> age; members of base class
cout << “ please enter the no. of workers under you:”;
cin >> now;
}
void manager : : show ( )
{
cout « "your name is : "«name«" and age is : "«age; cout «"In no. of workers under your are : "«now;
main ( )
{
clrscr ( ) ;
manager ml;
ml.get ( ) ;
cout « "\n \n";
ml.show ( );
}
```

Visibility modes

Basically we have visibility modes to specify that in which mode you are deriving the another class from the already existing base class. They are:

a. Private: when a base class is privately inherited by a derived class, 'public members' of the base class become private members of the derived class and therefore the public members of the base class can be accessed by its own objects using the dot operator. The result is that we have no member of base class that is accessible to the objects of the derived class.

b. Public: On the other hand, when the base class is publicly inherited, 'public members' of the base class become 'public members' of derived class and therefore they are accessible to the objects of the derived class.

c. Protected: C++ provides a third visibility modifier, protected, which serve a little purpose in the inheritance. A member declared as protected is accessible by the member functions within its class and any class immediately derived from it. It cannot be accessed by functions outside these two classes.

Visibility modifiers

The below mentioned table summarizes how the visibility of members undergo modifications when they are inherited

Base Class Visibility	Derived Class Visibility		
	Public	Private	Protected
Private	X	X	X
Public	Public	Private	Protected
Protected	Protected	Private	Protected

- The private and protected members of a class can be accessed by:

a. A function i.e. friend of a class.

b. A member function of a class that is the friend of the class. c. A member function of a derived class.

Student Activity

1. Define Inheritance. What is the inheritance mechanism in C++?

2. What are the advantage of Inheritance?

3. What should be the structure of a class when it has to be a base for other classes?

4. Can the derived class inherit static members?