CHAPTER 9

INHERITANCE

9.1 Introduction

Inheritance is the most powerful feature of an object oriented programming language. It is a process of creating new classes called **derived classes**, from the existing or **base classes**. The derived class inherits all the properties of the base class. It is a power packed class, as it can add additional attributes and methods and thus enhance its functionality. We are familiar with the term inheritance in real life (children acquire the features of their parents in addition to their own unique features). Similarly a class inherits properties from its base (parent) class .

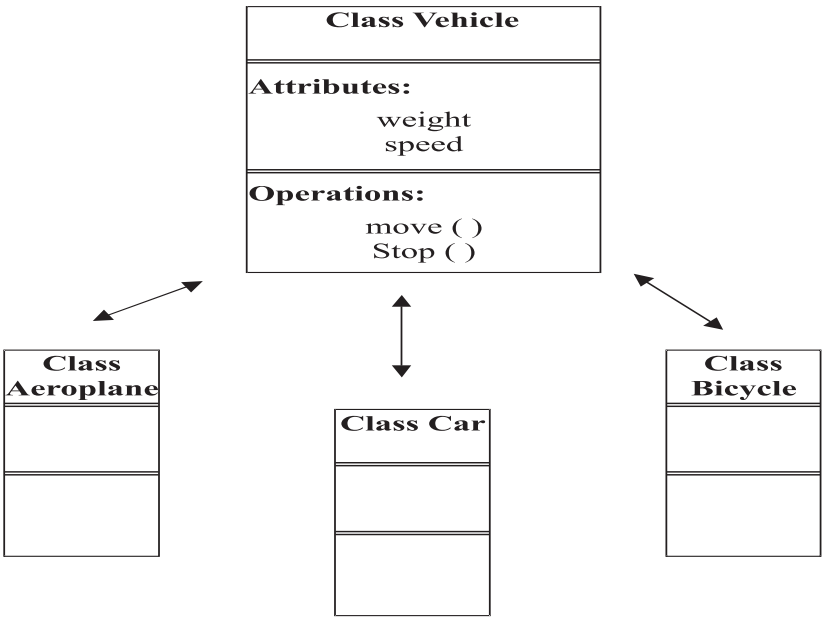


Fig.9.1 Inheritance

9.2 Advantages of inheritance

Inheritance has the following basic advantages.

1) Reusability of code : Many applications are developed in an organization. Code developed for one application can be reused in another application if such functionality is required. This saves a lot of development time.
2) Code sharing : The methods of the base class can be shared by the derived class.
3) Consistency of interface: The inherited attributes and methods provide a similar interface to the calling methods. In the Fig. 9.1 the attributes and methods of the class vehicle are common to the three derived classes – Aeroplane, Car and Bicycle. These three derived classes are said to be having a consistence interface.

9.3 Derived Class and Base class

A base class is a class from which other classes are derived. A derived class can inherit members of a base class.

While defining a derived class, the following points should be observed

a. The keyword **class** has to be used
b. The name of the derived class is to be given after the keyword class
c. A single colon
d. The type of derivation, namely **private, public or protected**
e. The name of the base class or parent class
f. The remainder of the derived class definition

```
// Program - 9.1

#include< iostream.h>
#include<conio.h>
class add
{
      int sum;
      protected:
            int num1, num2;
      public:
      add()
      {
         num1= num2= sum=0';
         cout<<"\n Add constructor .. ";
      }

      accept()
      {
         cout<<"\n Enter two numbers .. ";
         cin>>num1>>num2;
      }

      plus()
      {
            sum = num1 + num2;
            cout<<"\n The sum of two numbers is .. "<< sum;
      }
};

class subtract :public add
{
      int sub;
      public:
       subtract()
       {
         sub = 0;
         cout<<"\n Subtract constructor .. ";
       }
       minus()
       {
         add::accept();
         sub= num1-num2;
         cout<<"\n The difference of two numbers are ... "
              << sub;
       }
};
```

```
void main()
{
     subtract s;
     int choice = 0;
     cout<<"\n Enter your choice ";
     cout<<" \n1. Add..\n2. Subtract ..";
     cin>>choice;
     switch( choice )
     {
           case 1:
                 s.accept();
                 s.plus();
                 break;
           case 2:
                 s.minus();
                 break;
     }
}
```

In the Program – 9.1 the base class is add and the derived class is subtract. The derived class should be indicated as

class der_name : visibility mode base class-id

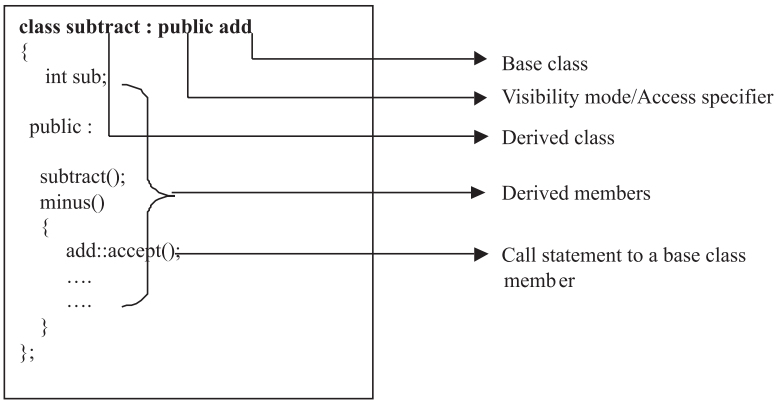
{

data members of the derived_class

functions members of derived_class

}

In the Program - 9.1 the derived class, subtract is defined as



9.4 Visibility Mode/Accessibility specifier

An important feature in Inheritance is to know as to when a member of a base class can be used by the objects or the members of the derived class. This is called as **accessibility**. The three access specifiers are private, protected and public. Access specifier is also referred to as visibility mode.The default visibility mode is private. The following Table 9.1 explains the scope and accessibility of the base members in the derived.

Base Class members	Derived Class		
	Private	Protected	Public
Private members	Are not inherited but they continue to exist		
Protected members	Inherits protected members as private members	Inherits protected and public as protected of derived	Protected members are retained as protected of the derived
Public members	Are inherited as private members of the derived	Inherits as protected members of the derived	Inherits public members as public of derived

Table 9.1 Scope and Access of Base Members in the Derived Class

When a base class is inherited with private visibility mode the public and protected members of the base class become 'private' members of the derived class

When a base class is inherited with protected visibility mode the protected and public members of the base class become ‘ protected members ‘ of the derived class

When a base class is inherited with public visibility mode , the protected members of the base class will be inherited as protected members of the derived class and the public members of the base class will be inherited as public members of the derived class.

> When classes are inherited publicly, protectedly or privately the private members of the base class are not inherited they are only visible i.e continue to exist in derived classes, and cannot be accessed

The declaration of classes add and subtract of Program-9.1 is as follows

```
Class add
{
     private:
          int sum;
    protected :
          int num1, num2;
   public:
          add();
          accept();
          plus();
};
class subtract : private add
{
      int sub;
   public:
      subtract();
      minus();
};
```

The data members and member functions inherited by subtract are:

int num1 & num2 with status as private of subtract

accept(); plus(); with status as private

I

```
class subtract : private add
{
       int sub;
  public:
      subtract( );
      minus();
};
```

II

```
class subtract : protected add
{
       int sub;
  public:
      subtract();
      minus();
};
```

III

```
class subtract : private add
{
       int sub;
  public:
      subtract();
      minus();
};
```

Accessibility of base members is shown in Table 9.2.

The constructors of the base class are not inherited, but are executed first when an instance of the derived class is created.

class subtract	II	III
private:	protected	public

sub
num1
num2
accept()
plus()
public:
subtract()
minus();
private mode
of inheritance
	sub
	protected:
	num1. num2
	accept();
	plus();
	public:
	subtract()
	minus()
	protected mode
	of inheritance
		sub
		protected:
		num1. num2
		public:
		accept();
		plus();
		subtract()
		minus();
		public mode
		of inheritance

Table 9.2 accessibility of Base Members

Consider the objects declared in the Program – 9.1. Complete the following table based on this program.

Constructors executed are _____ , _____

The objects of classes	DATA MEMBERS	METHODS/ FUNCTIONS
add		
subtract		

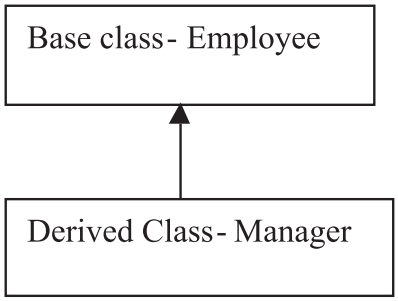
Table 9.3 Complete this Table

9.5 Types of inheritance

Classes can be derived from classes that are themselves derived. There are different types of inheritance viz., Single Inheritance, Multiple inheritance, Multilevel inheritance, hybrid inheritance and hierarchical inheritance.

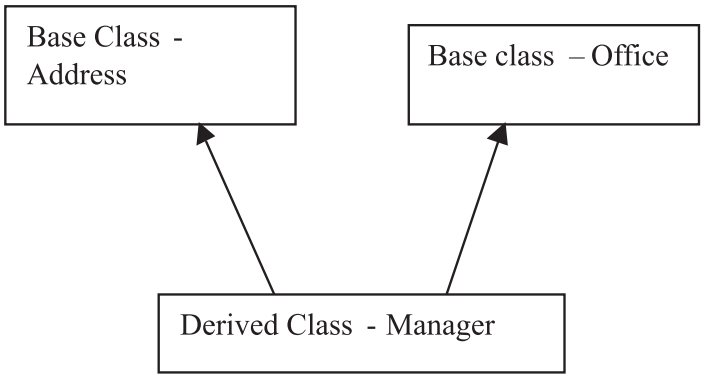
1) Single Inheritance

When a derived class inherits only from one base class, it is known as single inheritance



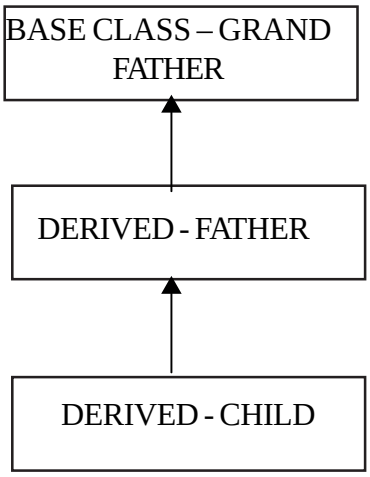
2) Multiple Inheritance

When a derived class inherits from multiple base classes it is known as multiple inheritance



3) Multilevel Inheritance

The transitive nature of inheritance is reflected by this form of inheritance. When a class is derived from a class which is a derived class itself – then this is referred to as multilevel inheritance.



What will be the output of Program 9.2?

```
// Program - 9.2
#include<iostream.h>
#include<conio.h>
class base
{
      public:
      base()
      {
            cout<<"\nConstructor of base class...
";
      }
      ~base()
      {
      cout<<"\nDestructor of base class.... ";
      }
};
class derived:public base
{
  public :
  derived()
  {
    cout << "\nConstructor of derived ...";
  }
  ~derived()
  {
      cout << "\nDestructor of derived ...";
  }
};

class derived2:public base
{
  public :
  derived()
  {
    cout << "\nConstructor of derived2 ...";
  }
  ~derived()
  {
      cout << "\nDestructor of derived2 ...";
  }
};

void main()
{
  derived2 x;
}
```

OUTPUT

Constructor of base class…

Constructor of derived ….

Constructor of derived2 …

Destructor of derived2…

Destructor of derived

Destructor of base class ..

✓ The constructors are executed in the order of inherited class i.e., from base constructor to derived. The destructors are executed in the reverse order

✓ Classes used only for deriving other classes are called as Abstract Classes ie., to say that objects for these classes are not declared.

Exercises

1) Given the following set of definitions

```
class node
{
        int x;
        void init();
    public:
        void read();
    protected:
        void get();
};
class type : public node
{
        int a;
public:
```

```
        void func1();
    protected:
        int b;
        void getb();
};
class statement :private type
{
        int p;
    public:
        void func2();
    protected:
        void func3();
};
```

Complete the following table

Members of the class type	Accessibility of members /their classes		
	private	protected	public
Members inherited by class type			
Defined in class type			

Table- 4 class node ....

Members of the class statement	Accessibility of members/ their classes		
	private	protected	public
Members inherited by class statement			
Defined in class statement			

Table- 5 class statement

Objects	Can access members	
	Data members	Member functions
class type		
class statement		

Table- 6 Objects…

2) Find errors in the following program. State reasons

```
#include<iostream.h>
class A
{
        private :
                int a1;
        public:
                int a2;
        protected:
                int a3;
};
class B : public A
{
        public:
        void func()
        {
                int b1, b2 , b3;
                b1 = a1;
                b2 = a2;
                b3 = a3;
        }
};
void main()
{
        B der;
        der.a3 = 0';
        der.func();
}
```

3) Consider the following declarations and answer the questions given below

```
class vehicle
{
        int wheels;
        public:
                void inputdata( int, int);
                void outputdata();
        protected :
                int passenger;
};
class heavy_vehicle : protected vehicle
{
        int diesel_petrol;
        protected:
                int load;
        public:
                void readdata( int, int);
                void writedata();
};
class bus: private _heavy_vehicle
{
        char marks[20];
        public:
                void fetchdata( char );
                void displaydata();
};
```

a. Name the base class and derived class of the class heavy_vehicle

b. Name the data members that can be accessed from the function displaydata()

c. Name the data members that can be accessed by an object of bus class

d. Is the member function output data accessible to the objects of heavy_vehicle class

4) What will be the output of the following program

```
#include<iostream.h>
#include<conio.h>
class student
{
        int m1, m2, total;
    public:
        student ( int a, int b)
        {
                m1 = a;
                m2 = b;
            cout<<"\n Non parameterized constructors..";
        };
```