INHERITANCE

PART- 7 (31)

(1) It is process of creating a new class, called derived class from existing class called base class.

(2) The advantage of inheritance is reusability.

(3) Reusing existing code saves time & money.

(4) Study of protected access specifier.

Access specifier	Accessible from own class	Accessible from derived class	Accessible from objects outside class
Public	Yes	Yes	Yes
Private	Yes	No	No
Protected	Yes	Yes	No

Format of derived class

```
class D: derivation B
{
    member of drive class;
}
```

derivation type of class →

(I) Public →
(II) Protected →
(III) Private →

(1) **Public Inheritance** → (I) Private member are not inherited Protected, Public both are inherited

(II) Public member of base class become public in derived class in public inheritance

(III) Protected member of " " " protected in derived class " " "

for eg sum of Number

```
class Base
{
Private : int a;
Protected: int b;
Public : int c;
void getdata();
void showdata();
};
```

```
class Derived : Public Base
{
Private : int d;
Public : void getd();
         void showd();
};
```

```
Void B:: getdata()
{
 cin >> a;
 cin >> b >> c;
}
Void B:: showdata()
{
 Cout << a << endl;
      << b << endl;
      << c << endl;
}
Void D:: getd()
{
 cin >> d;
}
Void D:: showd()
{
 int sum = b+c+d;
}
```

Note → a is private member of class Base and can't be inherited but object of derived class are able to access it through an inherited member function of Base i.e getdata & showdata.

```
Void main()
{
Derived d1;
d1. getdata();
d1. getd();
d1. showdata();
d1. showd();
getch();
}
```

Protected inheritance

→ (I) Public member of base class become protected in derived class

(II) Protected become protected

(III) Private are not inherited bt we can access private member through the inherited member fn of base class.

for eg

```
class Base
{
Private: int a;
Protected: int b;
Public: void getdata();
        void showdata();
};
class Derived: Protected B
{
Private: int c;
Public: void getd();
        Void showd();
};
void B :: getdata()
{
  cin>>a>>b;
};
void B :: showdata()
{
  cout<<a<<b;
}
Void D :: getd()
{
  cout cin>> c;
}
void D :: showd()
{
cout << c;
}
```

```
void showmain()
{
derived d1;
d1.getd();
d1.showd();
getch();
}
```

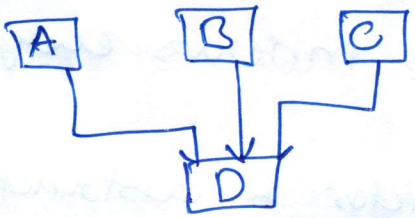
Private inheritance → (1) Public and protected both type of member of base class become private in derived class.

Derivation type	Type of base class: Public	Protected	Private
Public	Public	Protected	Not accessible
Protected	Protected	Protected	,,
Private	Private	Private	,,

Types of Inheritance →
1. Single
2. Multiple
3. Multilevel
4. Hierarchical
5. Hybrid

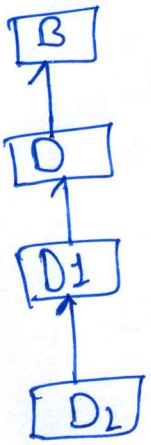
(II) Multiple Inheritance → More than One Base class

```
class D : derivation B1, derivation B2
{
member of class D;
}
```



WAP → make of Info/Biodata of student and Marks of student class from class and print complete Bio-data of student.

(III) Multilevel →

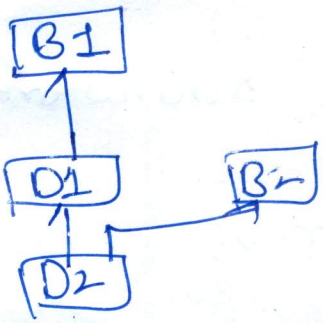



```
class B { };
class D: Public B
{
};
class D1: Public D
{
};
```

(iv) Hierarchical Inheritance →


```
class B{ };
class D : Public B{ };
class D1: Public B{ };
class D2: Public B{ };
```

(v) Hybrid Inheritance → More than one inheritance type in a single program.

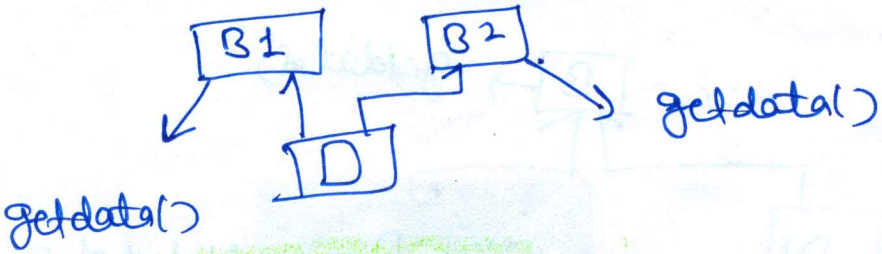


Note A derive class inherits every member of a base except

(i) its constructor & destructor
(ii) its friends
(iii) its operator =() members

Ambiguity in inheritance →

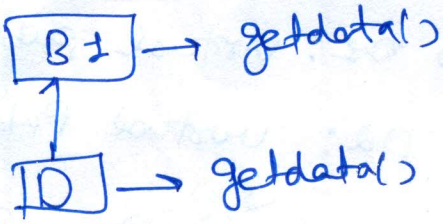
Case 1 →



Soln → can be solved by scope resolution operator i.e

```
void main()
{
  D obj;
Obj.B1::getdata();
Obj.B2::.getdata();
```

Case →

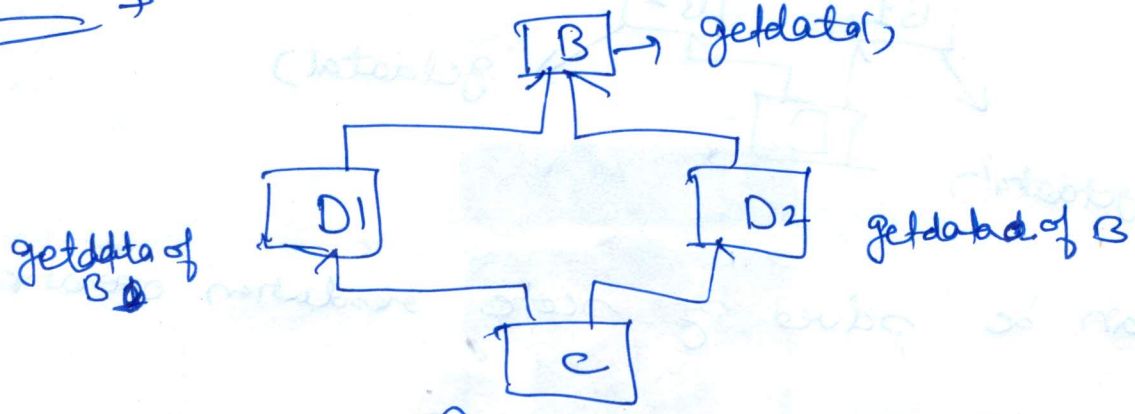


Sol →

```
void main()
{
  D obj;
  obj getdata(); // call to Derived class getdata();
  obj.B1::getdata(); // call to base class getdata();
```

Virtual Base Class

Problem →



(I) getdata from B to D1 & D1 to C
(II) getdata from B to D2 & D2 to C

Soln → (I) to remove ambiguity, common base class is declared as virtual Base class by writing virtual keyword.

```
class B { protected : int data; };

Prog →  class D1: virtual public B {};
        class D2: virtual Public B {};
        class c: Public D1, Public D2
        { public: int showdata();
          {
            return (data);
          }
        };
```

(II) It is used to share a single common inherited copy of data.

Abstract class

(I) It is a class that serves only as a base class from which classes are derived.

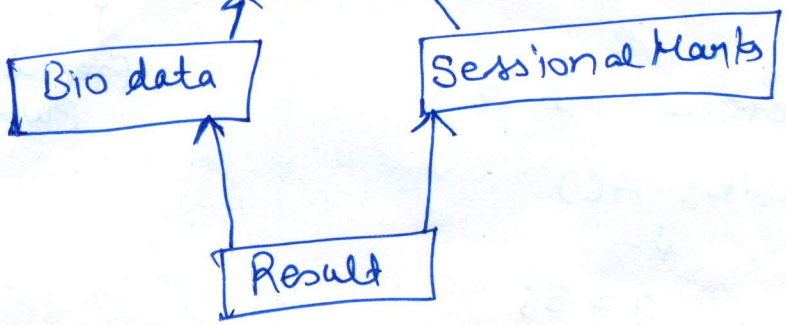
(II) No objects of an abstract class are created.

(III) A class that contains at least one pure virtual function is said to be abstract.

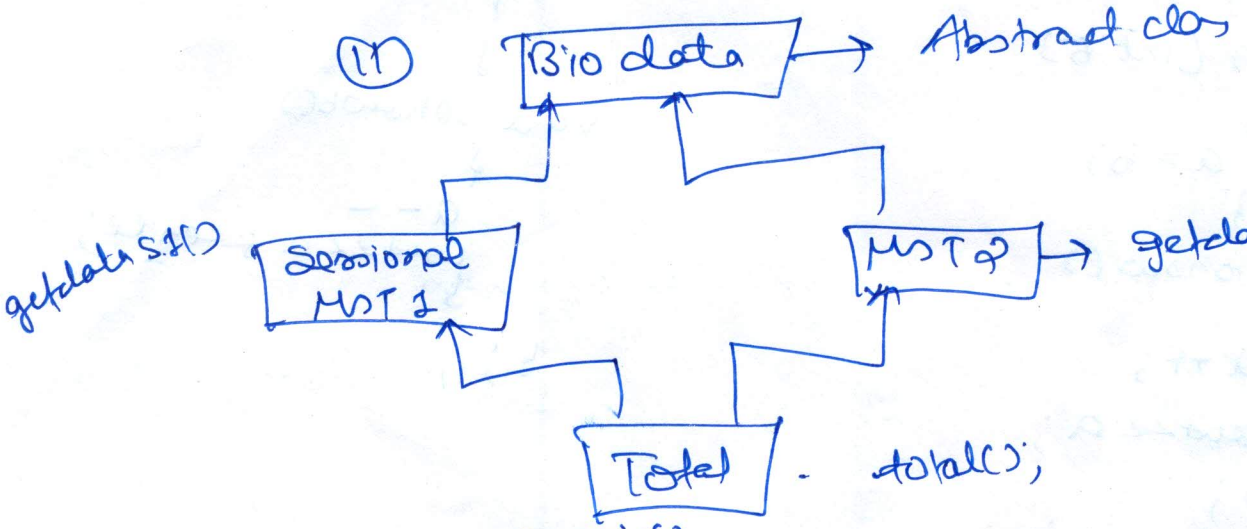
(IV) Although we can't create objects of an abstract class we can create pointers & references to an abstract class. which allows abstract classes to support run-time polymorphism.

for eg →

(I)



(II)




```
void main()
{ total t
  t.getdata();
  t.getdataS1();
  t.getdataS2();
  t.total();
}
```

Constructor in Inheritance

Note → (I) if there is no constructor specify in derived class then derived class will use the appropriate constructor (no arg constructor) of base class.

(II) The derived class does not call the more complex constructor i.e one argument, two argument if in derived class no ~~argument~~ constructor is specify.

(III) But if we want to call complex constructor we can do this by writing constructor in derived class i.e through derived class we can call base class constructor.

```
class A
{
protected: int a;
public: A()
    {
    a = 5;
    }
    A(int b)
    {
      a = b;
    }
void show()
  {
  a++;
  cout<<a;
  };
class void main()
  {
  B obj;        // a=5
  B obj1(10);   // a=10
  obj.showb();  // a=4
  obj.show();   // a=5
  obj1.showb(); // a=9
  } obj1.show(); // a=10
```

```
class B : Public A
{
protected:
  Public: B() : A()
    {
    }
  B(int b) : A(b)
    {
    }
  void showb()
    {
    a--
    cout<<a<<endl;
    }
};
```

(36) Order of execution of constructor & destructor in derived class

(I) In case of Multiple inheritance, when both the derived & base classes contains constructors, the base class const is executed first & then the constructor in the derived class is executed.

(II) In case of multiple inheritance, base classes are constructed in order in which they appear in the declaration of derived class.

(III) For eg →

(1)
```
class B : public A
{
};
```

(2)
```
class B : public A, Public C
{
};
```

(3)
```
class B : public A, virtual Public C
{
};
```

order of execution

(1) A() ⇒ base const^r
B() ⇒ derived const^r

(2) A() ⇒ Base(first)
C() ⇒ Base(second)
B() ⇒ derived

(3) C() ⇒ virtual Base
A() ⇒ Base
B() ⇒ derived class

(IV) Constructor of Base class can be called by derived class const^r by using initialization list;

nesting of class → we can create object of one class within the another class.

```
Class B1
{
Private: int a;
Public : void showB1()
        { a=10;
        cout<<a<<endl;
        }
};
Class B2
{
Private: int b;
Public: void showB2()
        { b=20
        cout<<b;
        }
};
Class D :
{
Prival: int c;
 B1 obj1;
 B2 obj2;
Public : void showD()
        {
         obj1.showB1(); // classes with classes
         obj2.showB2();
          c=30;
          cout<<c<<endl;
          }
};
void main()
 {
  D obj3;
  obj3.showD();
  getch();
}
```

// all object of D contain the objects obj1 & obj2 of B1 & B2

This relationship is called as containership.

Object composition & delegation

When a derived class object is assigned to Base class, the base class content in the derived class object are copied to the base class leaving behind the derived class speafy content. That is base class object can access only the base class members.

```
class base
{
Public: int i, j;
};
class derived : Public base
{
 Public: int k;
 };
int main()
{
  base b;
  derived d;
   b = d;      // Object slicing
  return 0;
}
```

b ← i, j

d ← i, j, k

⇓

b = d;

b ← i, j only