Object Oriented Programming Using C++

Ketan Kore
ketan.kore@sunbeaminfo.com

Association

- Consider following examples:
    1. Car has-a engine
    2. Room has-a wall
    3. Room has-a chair
    4. Employee has-a join date.

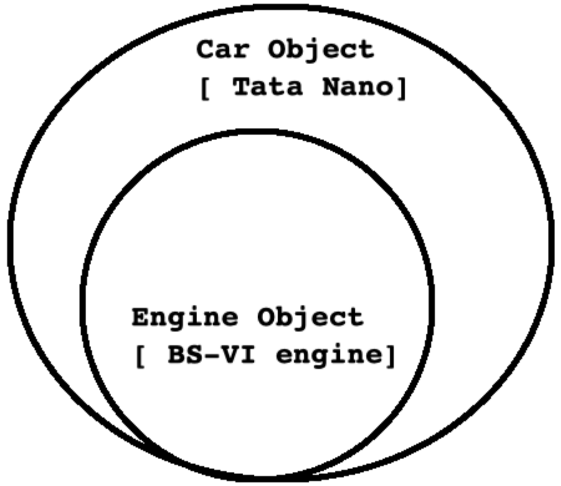
- If "has-a" relationship is exist between the types then we should use association.

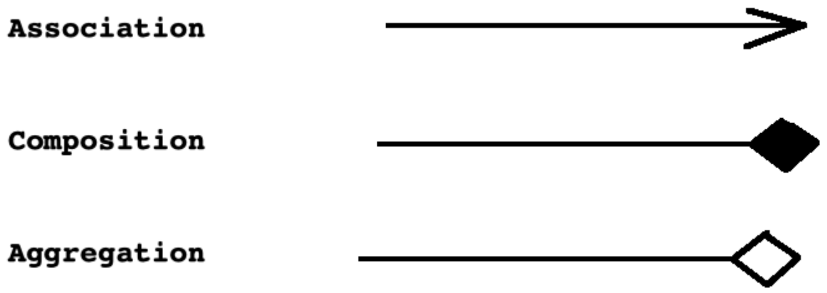
- Relation between car and engine can be considered as follows:
    1. Car has-a engine
    2. Engine is part of car

Association

Engine is part of car
  - BS-VI engine is a part of Tata Nano.

```
class Engine{
  //TODO
};

class Car{
  Engine e; //Associaition
  //TODO
};

Car c;
```





Association

- There are two special forms of Association:
    1. Composition
    2. Aggregation
- **Composition**

```
class Heart{
    //TODO
};
class Human{
    Heart heart;    //Association   =>  Composition
};
//Dependant Object  :   Human object
//Dependancy Object :   Heart Object
```

- In case of association, if dependency object do not exist without Dependant object then it is called composition.
- In other words, composition represents tight coupling.

Association

- **Aggregation**
  - Let us consider example of department and faculty

```
class Faculty{
};
class Department{
    Faculty faculty;    //Associaition  :   Aggregation
};
//Dependant Object  :   Department object
//Dependancy Object :   Faculty Object
```

  - In case of association, if dependency object exist without dependant object then it is called aggregation.
  - In other words, aggregation represents loose coupling.

Inheritance

- If "is-a" relationship is exist between the types then we should use inheritance.
- **Inheritance is also called as generalization.**
- Example:
    1. Employee is a Person
    2. Manager is a Employee
    3. Book is a product
    4. Circle is a Shape
    5. SavingAccount is a account.
    6. Car is a vehicle

Inheritance

- Let us consider example of employee and person:
  - ➢ Employee is a Person

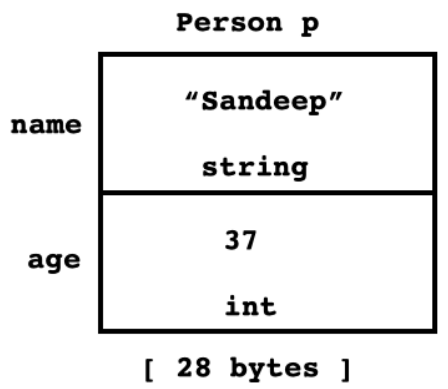
```
class Person{   //Parent class
    //TODO
};
class Employee : public Person{ //Child class
    //TODO
};
```

- In C++, parent class is called base class and child class is called derived class.
- In above statement, "public" is called as mode of inheritance. It can be private/protected/public.
- **In C++, default mode of inheritance is private.**

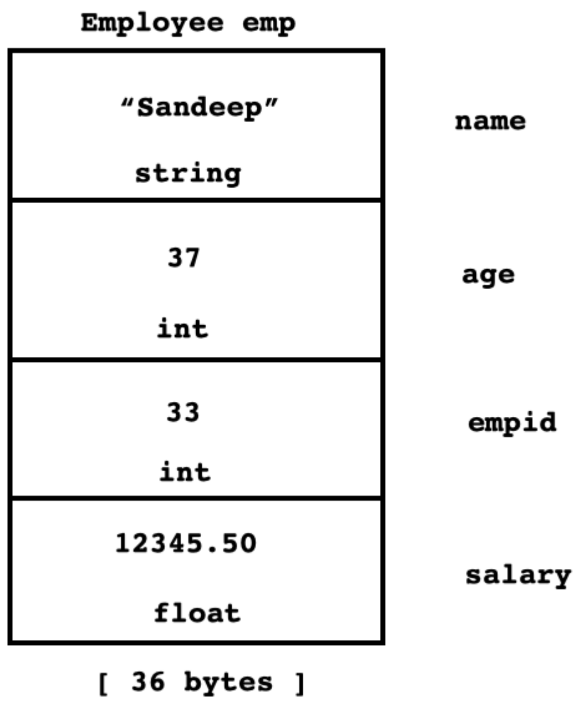
Inheritance

- During inheritance, members(data member / member function / nested type) of derived class, do not inherit into base class. Rather members of base class, inherit into derived class.
- All the non static data members of base class get space inside object of derived class. In other words, **non static data members of base class, inherit into derived class**.
- Using derived class, we can access static members of base class. In other words, **static data members of base class, inherit into derived class**.
- All the data members(static/non static) of base class of any access specifier(private/protected/public), inherit into derived class but **only non static data member get space inside object.**
- Size of object of Base class = size of of all the non static data members declared in base class.
- **Size of object of Derived class = size of of all the non static data members declared in base class + size of of all the non static data members declared in derived class.**

Inheritance



```
Person p(“sandeep”, 37);

p.showRecord( );
```



```
Employee emp;

emp.showRecord( );

emp.displayRecord( );
```

Inheritance

- We can call, non static member function of base class on object of derived class. In other words, non static member function inherit into derived class.

- We can call static member function of base class on derived class. In other words, static member function inherit into derived class.

- **Following function do not inherit into derived class:**
    1. **Constructor**
    2. **Destructor**
    3. **Copy constructor**
    4. **Assignment operator function**
    5. **Friend function**

- Except above 5 functions, all the member functions(static/non static) of base class inherit into derived class.

Inheritance

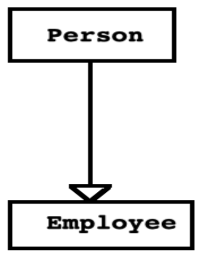
- If we create object of base class then only base class constructor and destructor gets called.
- **If we create object of derived class then first base class constructor gets called and then derived class constructor gets called. Destructor calling sequence is exactly opposite.**
- From any constructor of derived class, by default, base class's parameterless constructor gets called.
- **If we want to call, any constructor of base class from constructor of derived class then we should use constructor's base initializer list.**

Inheritance

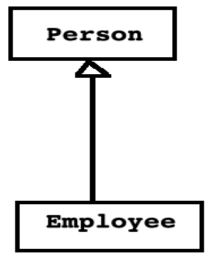
```
class Base{
    int num1;
public:
    Base( int num1 ){
        this->num1 = num1;
    }
};
class Derived  : public Base {
    int num2;
public:
    Derived( int num1, int num2 ) : Base( num1 ) //constructor's base initializer list {
        this->num2 = num2;
    }
};
Derived d( 100, 200 );
```

Inheritance

- How should we read following statement?
  - **class Employee : public Person**
    1. Class Person is inherited into class Employee.
    2. Class Employee is derived from class Person.

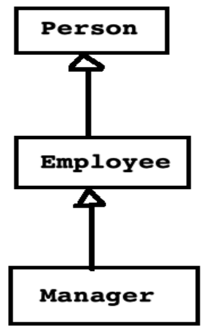


Class Person is
Inherited into
Class Employee.



Class Employee is
Derived from
Class Person.

[Recommended]



Class Employee is direct Base class
And class Person is indirect Base class
Of class Manager.

Class Employee is direct Derived class
And class Manager is indirect derived class
Of class Person.

Inheritance

- If we use private/protected/public keyword to control visibility of members of class then it is called access specifier.

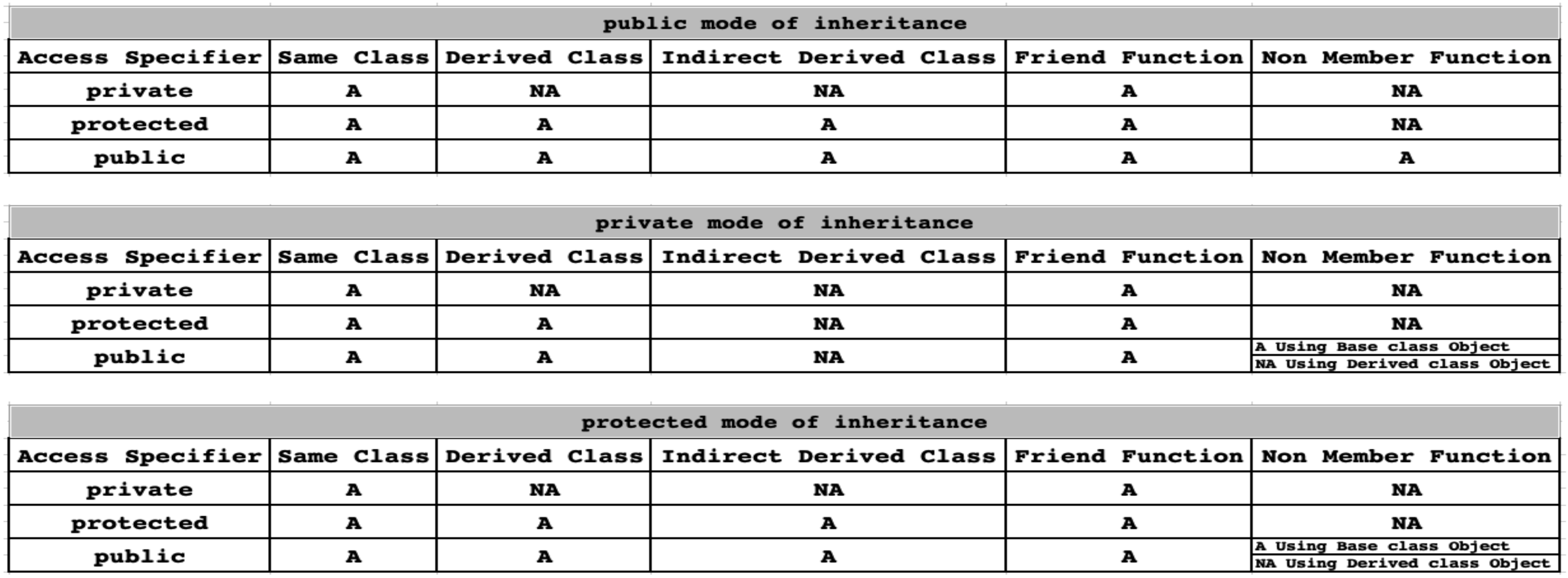
- If we use private/protected/public keyword to extend the class / to create derived class then it is called mode of inheritance.

  1. **private mode(default mode)**
     - ➢ If has-a relationship is exist between the types then we should use either association or private mode of inheritance.

  2. public mode
     - ➢ If is a relationship is exist between the types then we should use public mode of inheritance.

  3. protected mode

Inheritance



public mode of inheritance					
Access Specifier	Same Class	Derived Class	Indirect Derived Class	Friend Function	Non Member Function
private	A	NA	NA	A	NA
protected	A	A	A	A	NA
public	A	A	A	A	A

private mode of inheritance					
Access Specifier	Same Class	Derived Class	Indirect Derived Class	Friend Function	Non Member Function
private	A	NA	NA	A	NA
protected	A	A	NA	A	NA
public	A	A	NA	A	A Using Base class Object / NA Using Derived class Object

protected mode of inheritance					
Access Specifier	Same Class	Derived Class	Indirect Derived Class	Friend Function	Non Member Function
private	A	NA	NA	A	NA
protected	A	A	A	A	NA
public	A	A	A	A	A Using Base class Object / NA Using Derived class Object

Inheritance

- During inheritance, if base type and derived type is interface then it is called interface inheritance.

- During inheritance, if base type and derived type is class then it is called implementation inheritance.

- **Types of inheritance**
  - **Interface Inheritance**
    1. Single Inheritance
    2. Multiple Inheritance
    3. Hierarchical Inheritance
    4. Multilevel Inheritance
  - **Implementation Inheritance**
    1. Single Inheritance
    2. Multiple Inheritance
    3. Hierarchical Inheritance
    4. Multilevel Inheritance

Inheritance

- **Single Inheritance**
  - Consider example:
    - Car is a Vehicle.

```
class Vehicle{   };
class Car : public Vehicle{ };
```

  - If single base class is having single derived class then it is called single inheritance.

Inheritance

- **Multiple Inheritance**
  - Consider example:
    - Allrounder is Baller and Batsman.

```
class Baller{    };

class Batsman{   };

class AllRounder : public Baller, public Batsman{    };
```

  - If multiple base classes are having single derived class then it is called multiple inheritance.

Inheritance

- **Hierarchical Inheritance**

  - Consider example:
    - Rectangle is a Shape
    - Circle is a Shape
    - Triangle is a Shape

```
class Shape{     };
class Rectangle : public Shape{ };
class Circle : public Shape{ };
class Triangle : public Shape{ };
```

  - If single base class is having multiple derived classes then it is called hierarchical inheritance.

Inheritance

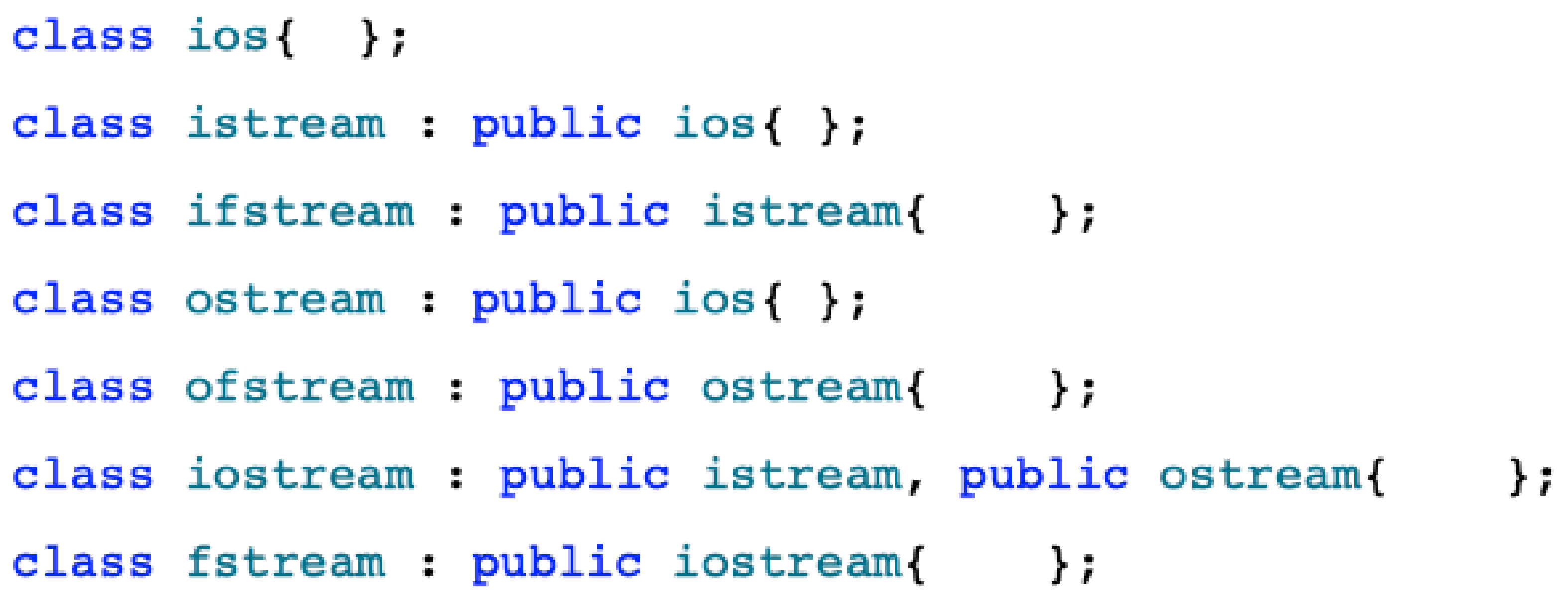
- **Multilevel Inheritance**
  - Consider example:
    - Employee is a Person
    - Manager is a Employee
    - SalesManager is a Manager

```
class Person{    };

class Employee : public Person{ };

class Manager : public Employee{ };

class SalesManager : public Manager{ };
```

  - If single inheritance is having multiple levels then it is called multilevel inheritance.

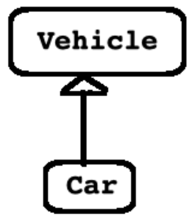
Inheritance

- **Hybrid Inheritance**
    - Combination of any two or more than two types of inheritance is called hybrid inheritance.

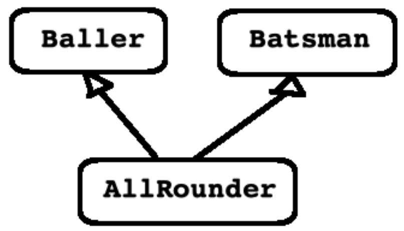


```
class ios{   };
class istream : public ios{ };
class ifstream : public istream{     };
class ostream : public ios{ };
class ofstream : public ostream{     };
class iostream : public istream, public ostream{     };
class fstream : public iostream{     };
```

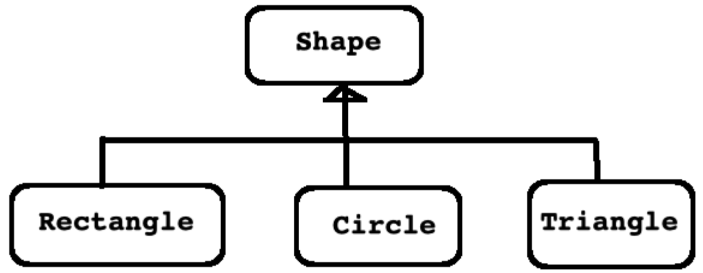
Inheritance



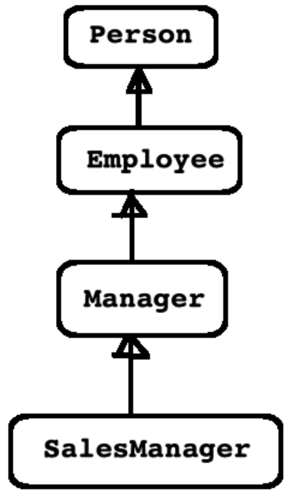
Single Inheritance



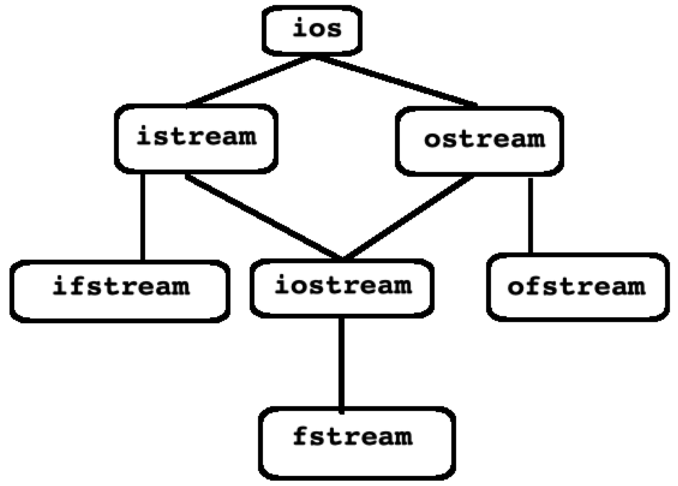
Multiple Inheritance



Hierarchical Inheritance



Multilevel Inheritance



Hybrid Inheritance

Inheritance

- If implementation of base class member function is logically incomplete then we should redefine member function inside derived class. In other words, we should give same name to the member function in derived class.

- If name of base class and derived class member function is same and if we try to call such member function on object of derived class then preference is given to the derived class member function. Here derived class member function hides implementation of inherited function. This process is called shadowing.

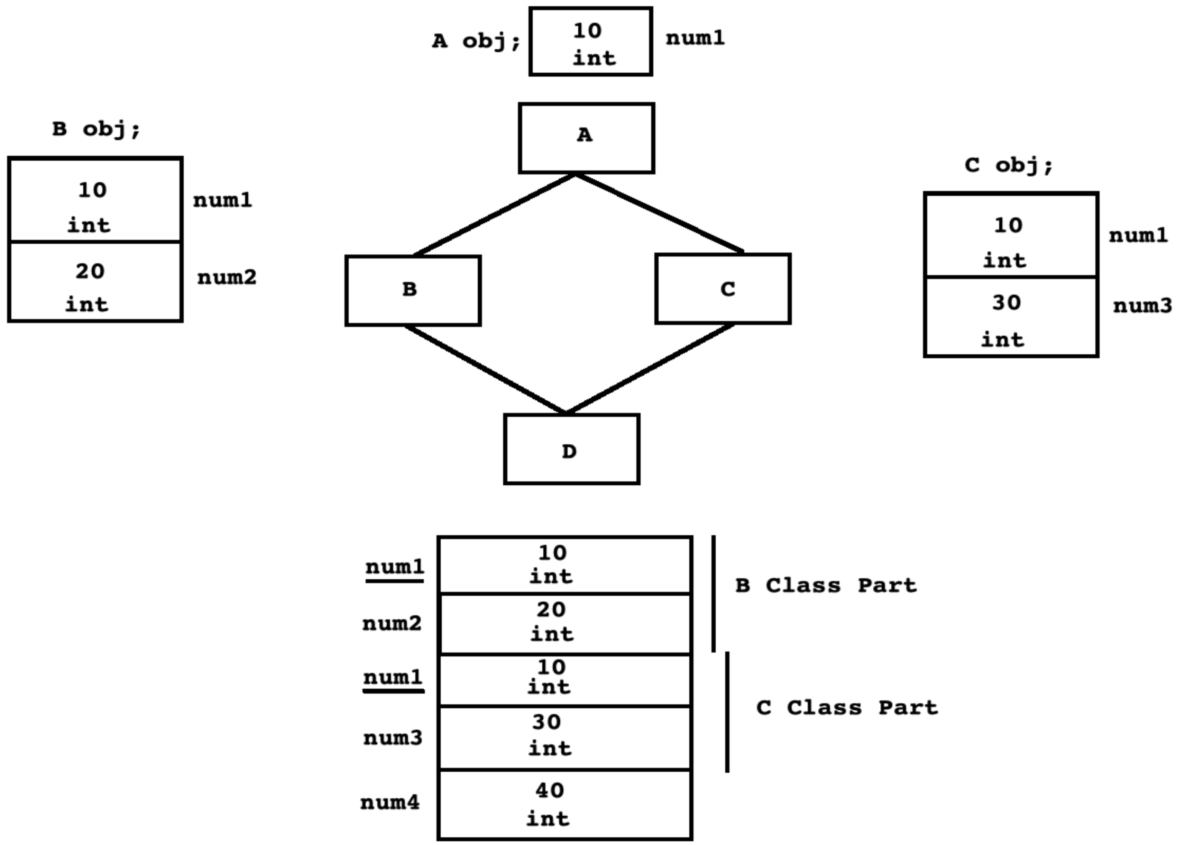
- Without changing implementation of existing class, if we want to extend meaning of that class then we should use inheritance.

Inheritance

```
class Person{
public:
    void print( void ){
        cout<<"Person::print"<<endl;
    }
};
class Employee : public Person{
public:
    void print( void ){
        cout<<"Employee::print"<<endl;
    }
};
int main( void ){
    Employee emp;
    emp.Person::print( );        //Person::print
    emp.Employee::print( );    //Employee::print
    emp.print( );                //Employee::print
    return 0;
}
```

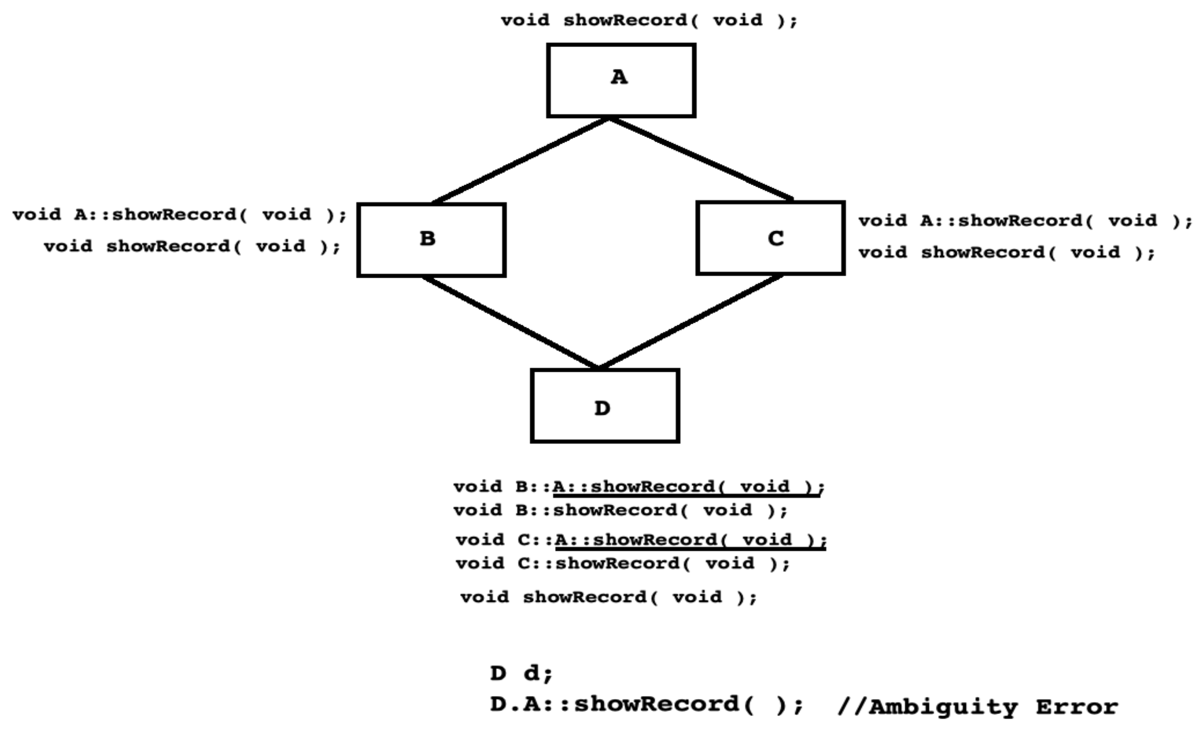
Inheritance

- **Diamond Problem : 1**



Inheritance

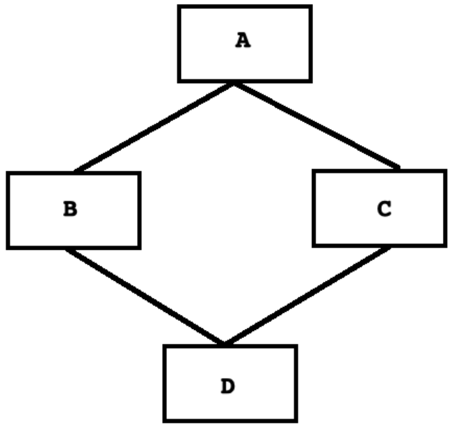
- **Diamond Problem : 2**


void showRecord(void);
A
void A::showRecord(void);
void showRecord(void);
B
C
void A::showRecord(void);
void showRecord(void);
D
void B::A::showRecord(void);
void B::showRecord(void);
void C::A::showRecord(void);
void C::showRecord(void);
void showRecord(void);

```
D d;
D.A::showRecord( );  //Ambiguity Error
```

Inheritance

- **Diamond Problem : 3**



```
A Class Object
- Constructor Call  : A( )
- Destructor Call   : ~A( )

B Class Object
- Constructor Call  : A( ) -> B( )
- Destructor Call   : ~B( ) -> ~A( )

C Class Object
- Constructor Call  : A( ) -> C( )
- Destructor Call   : ~C( ) -> ~A( )

D Class Object
- Constructor Call  : A( ) -> B( ) -> A( ) -> C( ) -> D( )
- Destructor Call   : ~D( ) -> ~C( ) -> ~A( ) -> ~B( ) -> ~A( )
```

Inheritance

- Let us say Diamond Inheritance = Hierarchical Inheritance + Multiple Inheritance.

- Problems occur during diamond/hybrid inheritance:
    1. During diamond inheritance, data members of indirect base class inherit into indirect derived class multiple times. Hence it affects on size of object of indirect derived class.

    2. During diamond inheritance, member function of indirect base class inherit into indirect derived class multiple times. If we try to call member function of indirect base class on object of indirect derived class, compiler generates ambiguity error.

    3. During diamond inheritance, if we create object of indirect derived class the constructor and destructor of indirect base class gets called multiple times.

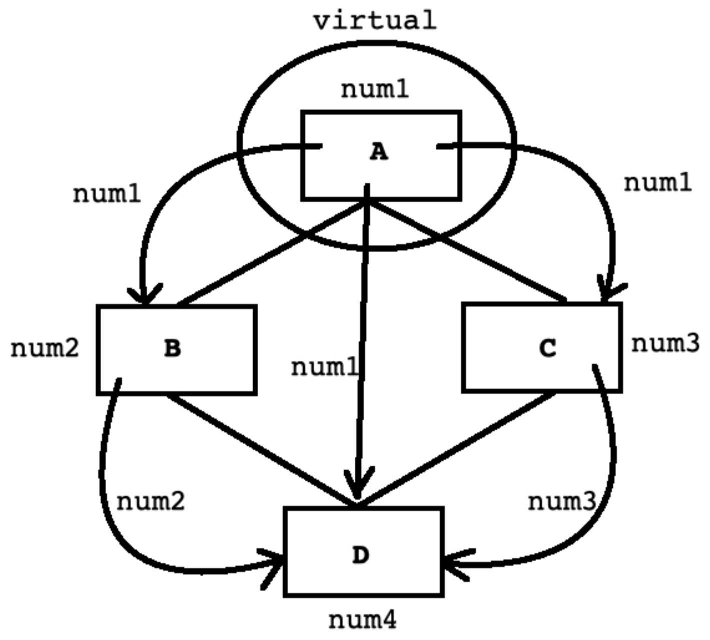
- **All above problems created by hybrid inheritance is called diamond problem**

Inheritance

- virtual is keyword in C++.
- **If we want to avoid diamond problem then we should declare base class virtual.**

```
class A{    };
class B : virtual public A{ };      //Virtual Inheritance
class C : virtual public A{ };      //Virtual Inheritance
class D : public B, public C{ };
```

Inheritance



Thank you