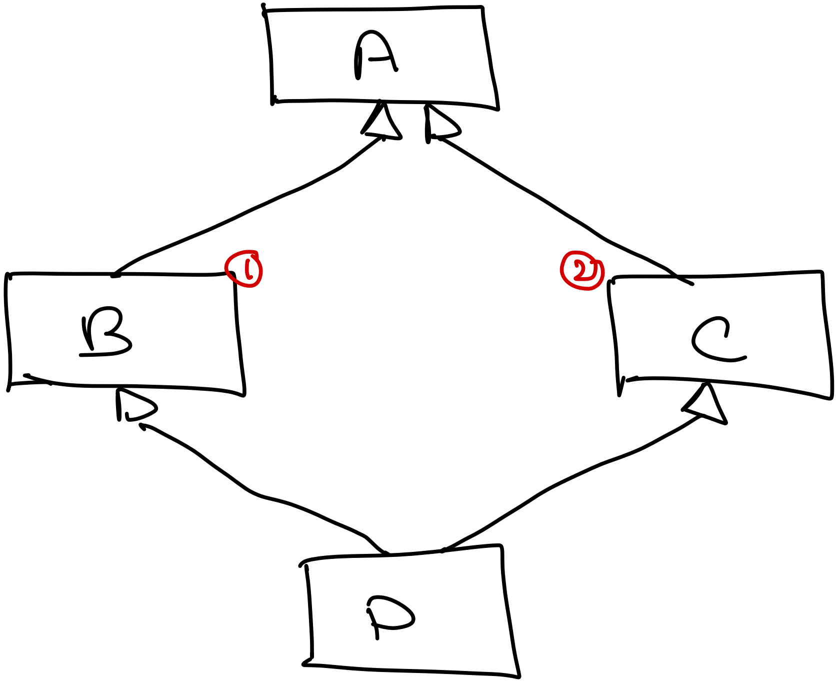


D obj;

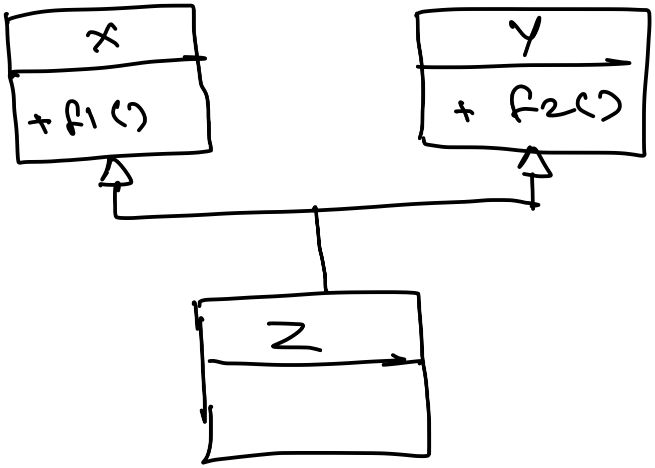
A

B

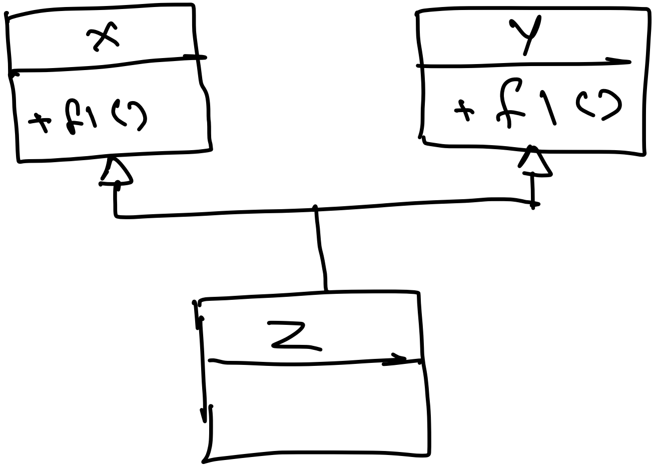
A

C

D



```
Z obj;
obj.f1();
obj.f2();
```



```
class Z : public X, public Y {
};
```

```
obj.X::f1();   X::f1()
obj.Y::f1();   Y::f1()
```

```
Z obj;
obj.f1();
```

error

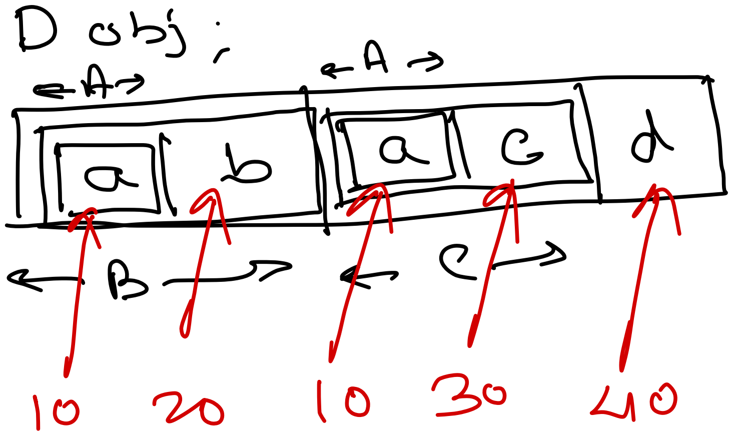
f1 is ambiguous

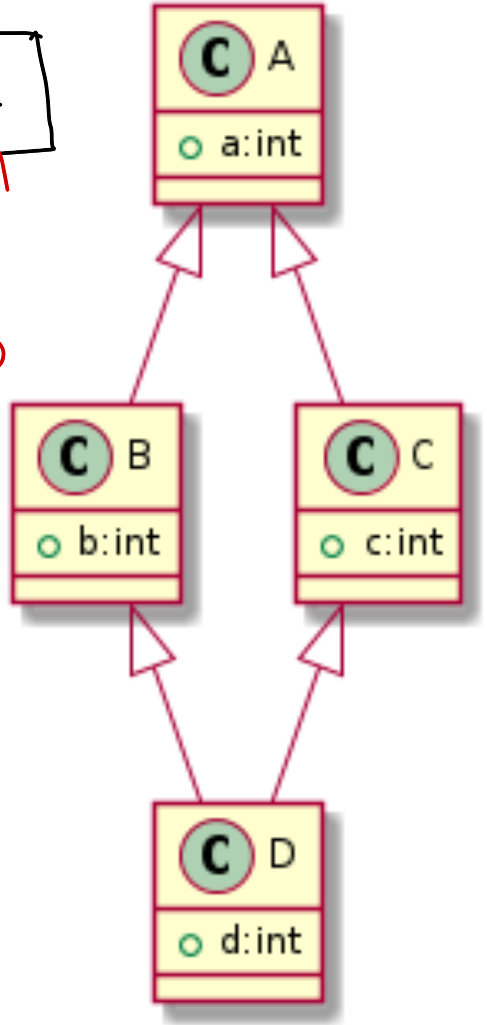
Diamond inheritance

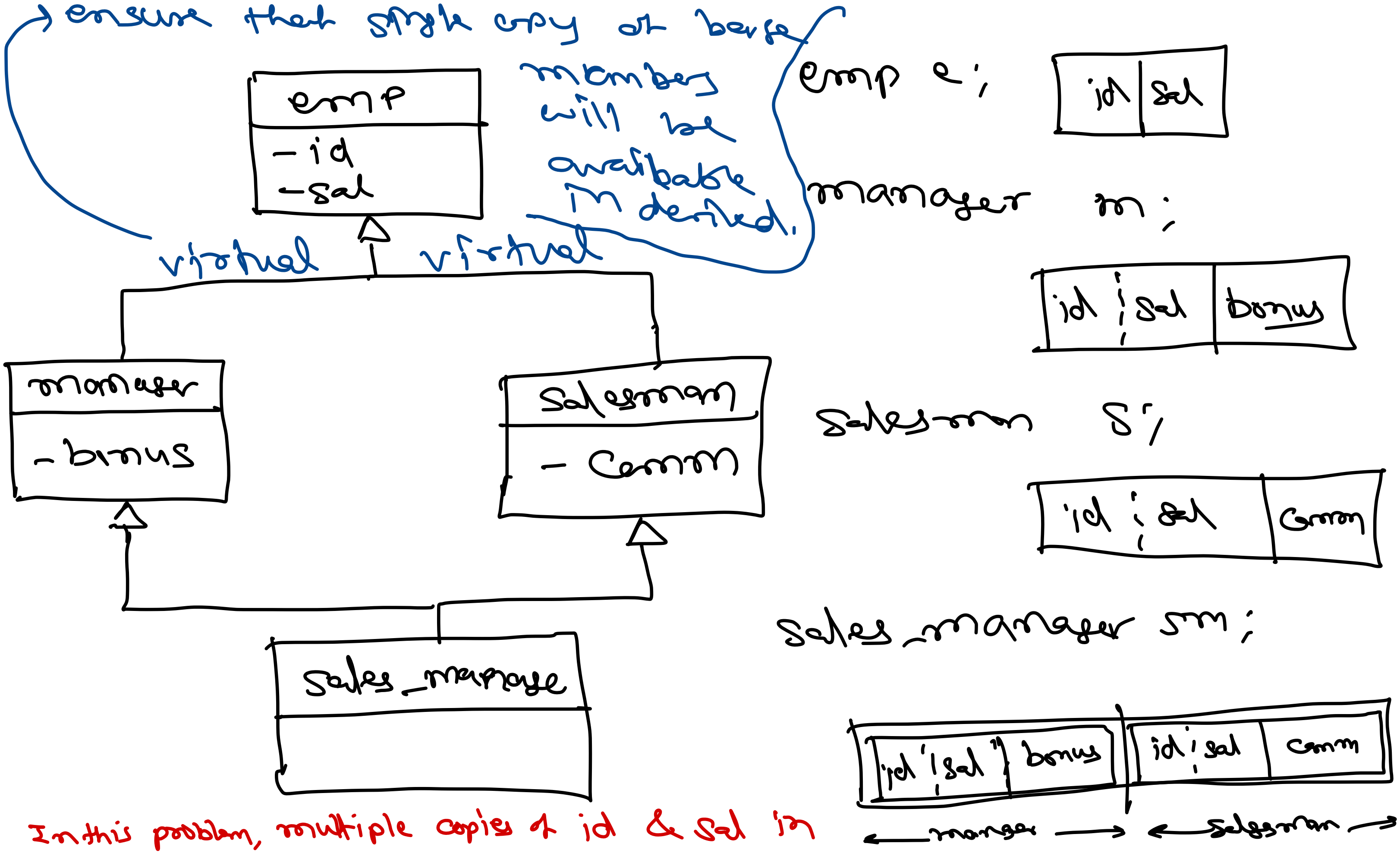
- If base class members are inherited into derived class via multiple inheritance path, multiple copies of those base members will be present in derived class.

Diamond Problem

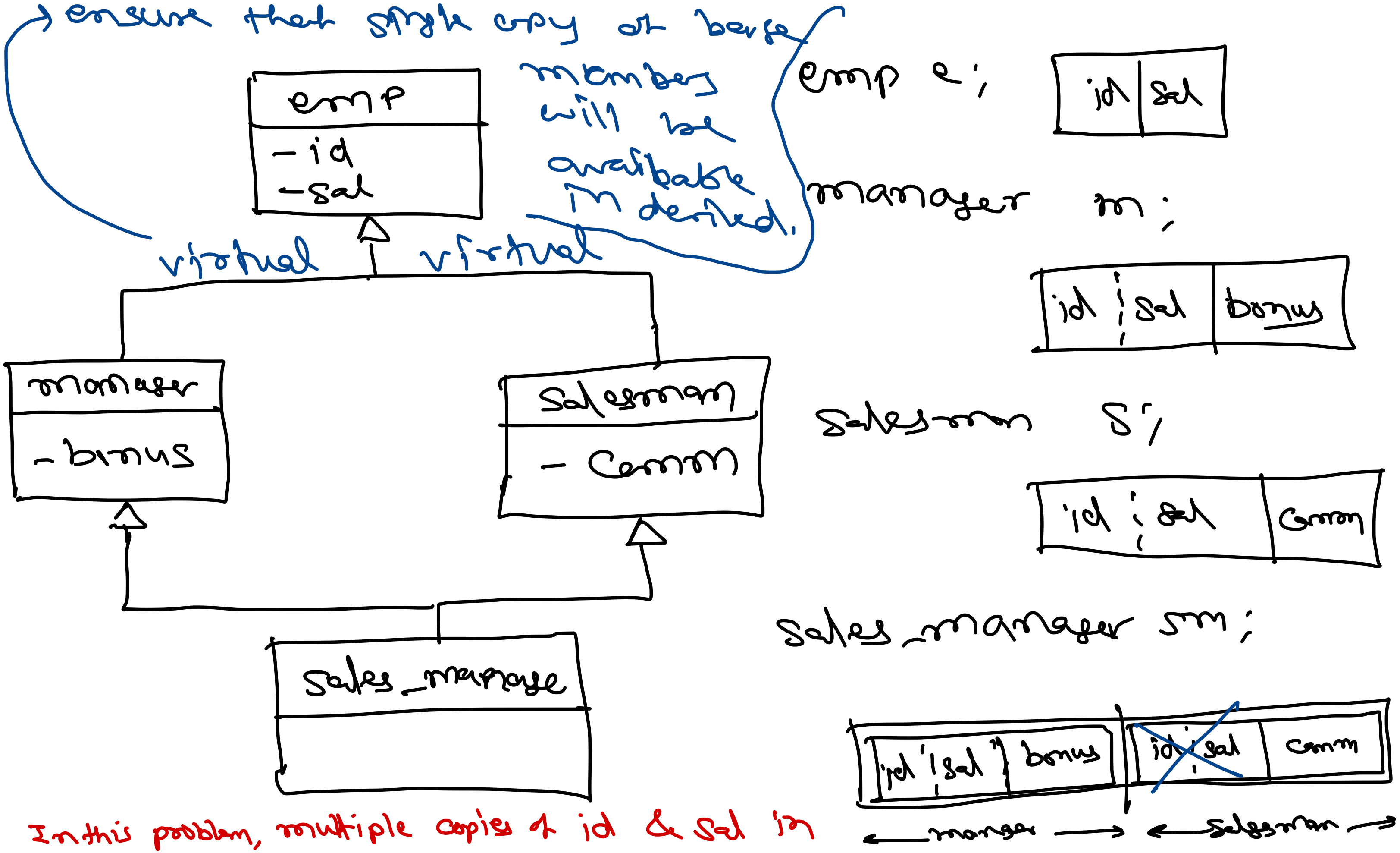
- Accessing such members from derived class object will lead to ambiguity error.
- It can be resolved using scope resolution operator.
- sizeof(D) = ?? 20
- Constructor calling sequence
  - A, B, A, C, D.
- Destructor calling sequence will be reverse.







In this problem, multiple copies of id & sal in sales_manager class are not desirable.



ensure that single copy of base member will be available in derived.

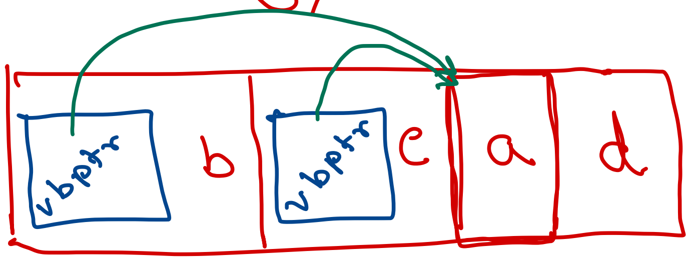
```
emp e;
manager m;
salesman s;
sales_manager sm;
```

In this problem, multiple copies of id & sal in sales_manager class are not desirable.

Diamond inheritance (Virtual)

- To ensure that base class members are not duplicated in derived class, it must be inherited virtually in immediate derived class.
- class A {...};
- class B: public virtual A {...};
- class C: public virtual A {...};
- class D: public B, public C {...};
- sizeof(D) = ??
- Constructor calling sequence
  - A, B, C, D.
- Destructor calling sequence will be reverse.

D obj;



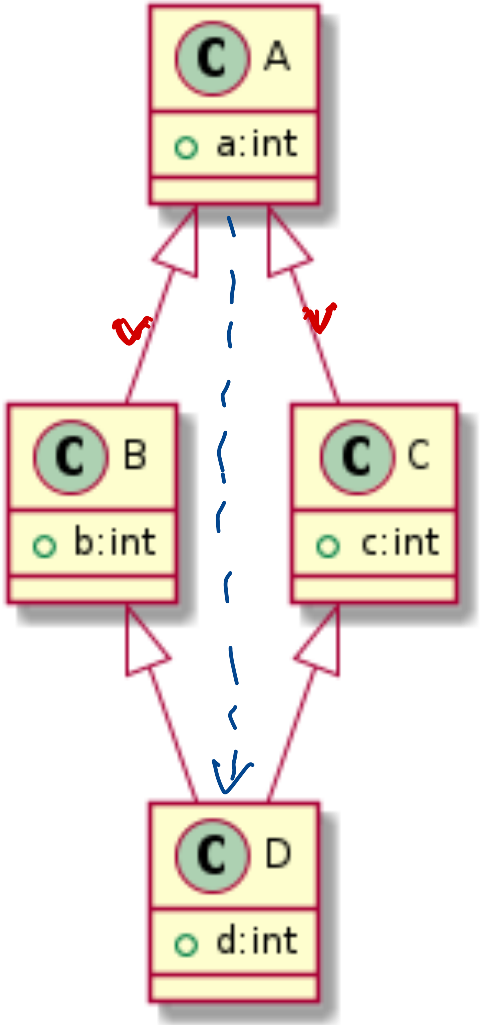
8 + 4 + 8 + 4 + 4 + 4
= 32

B obj;

vbptr	b	a

A obj;

a



Modes of inheritance

class A : public B { };
protected
private

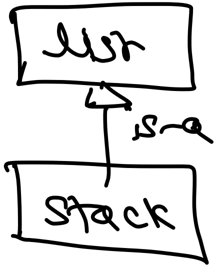
- C++ has three modes of inheritance
  - Private
  - Protected
  - Public
- Private members of a class are never accessible outside the class or in inherited class.
- Visibility of a member in derived class depends on its access in base class and mode of inheritance.
- Visibility of member can be altered by redeclaring member in the derived class.

inheritance modes → In base → access ↓	public	protected	private
public	public	protected	private
protected	protected	protected	private
private	private (N.A)	private (N.A)	private (N.A)

stack is a linked list, in which add & delete is done from same end to achieve LIFO behavior.
e.g. add_first() → push(), del_first() → pop().

```
class list {
    —
    —
public:
    add_first();
    add_last();
    del_first();
    del_last();
    —
    —
};
```

```
                    ~~public~~
class stack : private list {
public:
    list::add_first;
    list::del_first;
};
```



redeclaring members of base into derived, to make them public again;

method hiding

```
class A {
public:
    void fun(int a){
      =
    }
};

class B: public A {
 public:
    void fun() {
      =
    }


};
```

```
main() {
    B obj;
    obj.fun(1);

}
```

Method shadowing

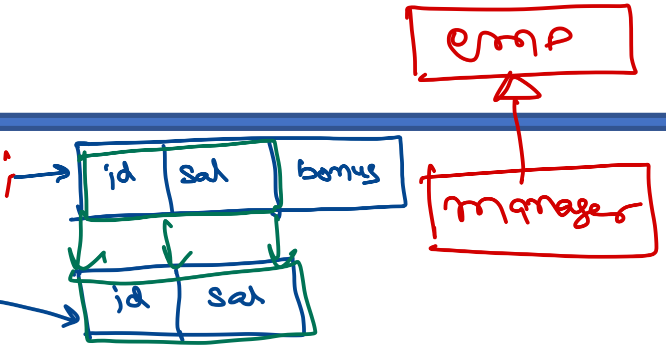
- Method in derived class hides all methods in base class having same name (with same or different signature).
- Using derived class object/pointer such hidden method (from base class) cannot be called.
- Such method can be called explicitly using scope resolution.
- Or such methods can be made visible in sub class by redeclaring it into derived class.

```
B obj;
obj.A::fun(1);

class A {
public:
    void fun(int a){
    }
};

class B: public A {
public:
    void fun() {
    }
    A::fun;
};
```

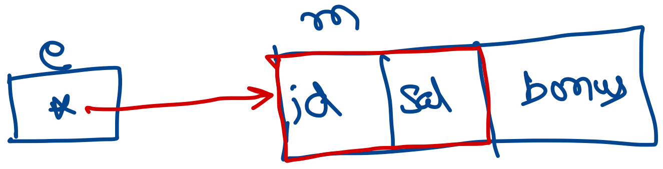
Object slicing

- If derived class object is assigned to base class object, only base class part present in the derived class object is assigned to the base class object.
- If derived class object's address is assigned to base class pointer, only address of base class part present in the derived class object is assigned to that base class pointer.
- With such pointer/reference only base class members of derived class object can be accessed.

```
manager m;
emp e;
e = m;
e.display();  →  emp::display(){ }
```



```
manager m;
emp *e;
e = &m;
e -> display();  →  emp::display()
                    {}

cout << e->get_id();
cout << e->get_sal();
cout << e->get_bonus();
→ error
```



Virtual function

- Virtual function is declared with virtual keyword in C++. → declaration of member function.
- If behaviour of any function is expected to be different than base class function, then that function is declared as virtual in base class and redefined in derived class with same signature.
- Redefining method in derived class with same signature is called as "function overriding".
- Virtual function is always called depending on type of object (not on type of caller).

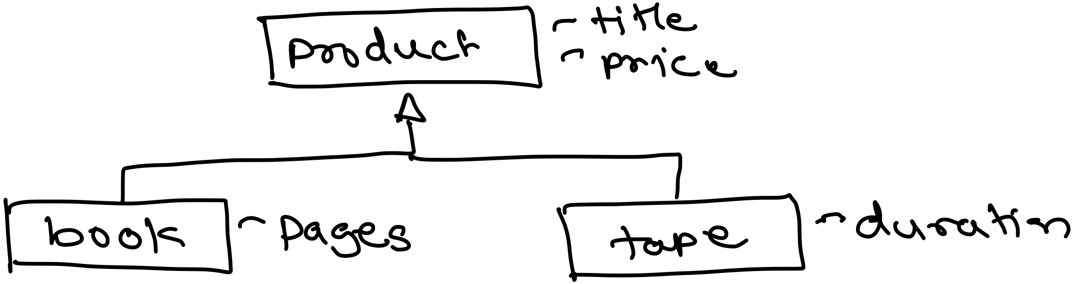
→ object
→ pointer
→ reference

Method overloading vs Method overriding

runtime polymorphism

- Methods with same name and different arguments. → in same scope.
- Based on arguments each method will be given different name internally by the compiler, called as "Name mangling". → false polymorphism
- Method to be called is decided by the compiler, depending on arguments. It is also referred as "Early binding".
- Return type of method is not considered (because collecting retturn value is not compulsory).
- Constructors can be overloaded, but destructors cannot be overloaded.

- Method with same prototype redefined in derived class.
- Internally use virtual function table for the class and virtual table pointer for the object, to invoke appropriate method at runtime. → true polymorphism.
- Method to be called is decided at runtime, depending on type of object. It is referred as "Late binding".
- Return type of method in derived class must be same or sub-type of return type as in base class.
- Constructor cannot be virtual, but destructor can be virtual (for special cases).


Product
~ title
~ price
book
~ pages
tape
~ duration

Abstract class

- If class needs a method which doesn’t have implementation (or partial), then it is declared as “pure virtual” or “abstract” method.
- In C++, pure virtual function declared as
  - virtual void func()=0;
  - 0 indicate NULL entry in vtable.
- Virtual function may not have body.
- Pure virtual fn must be overridden in the derived class; otherwise it remains abstract.
- If class contains atleast one function as pure virtual, then object of such class cannot be created & is referred as “abstract class”.