Sunbeam Infotech

Exploring new ideas, Reaching new heights!

A

B

A

C

D

D obj;

Left diagram:

```
     X                    Y
  + f1()              + f2()
         \          /
            Z
```

```
Z obj;
obj.f1();
obj.f2();
```
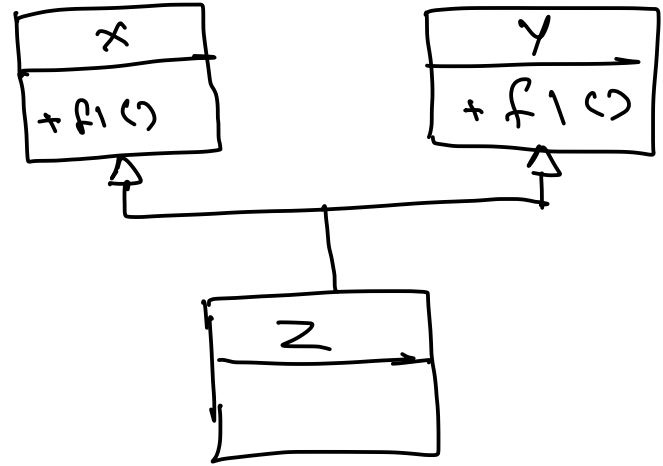
Right diagram:

```
     X                    Y
  + f1()              + f1()
         \          /
            Z
```
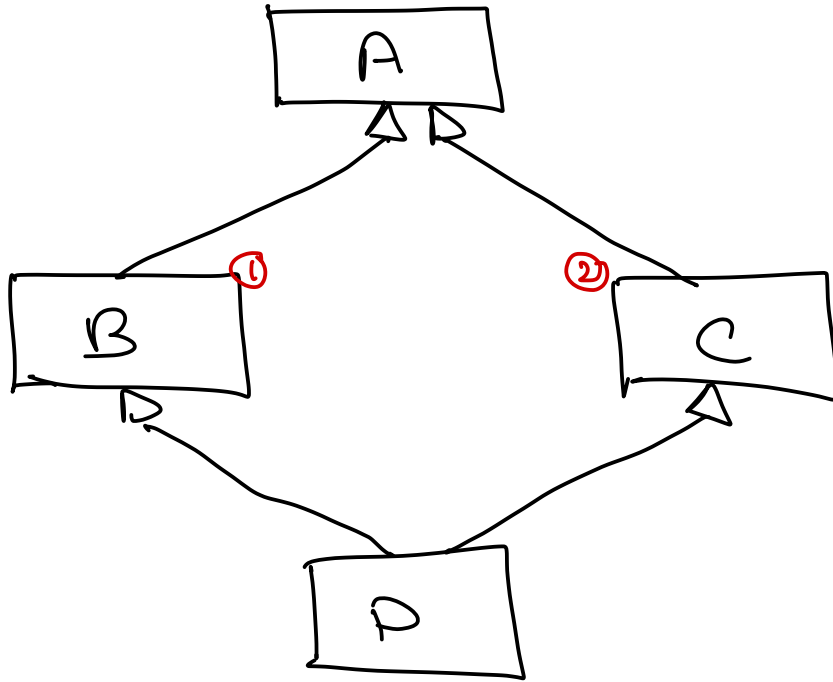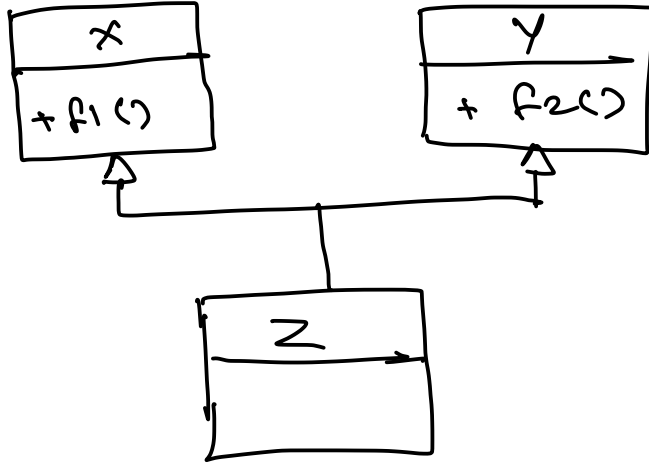
```
class Z: public X, public Y {
};
```

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

                       → error
Z obj;
obj.f1();              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

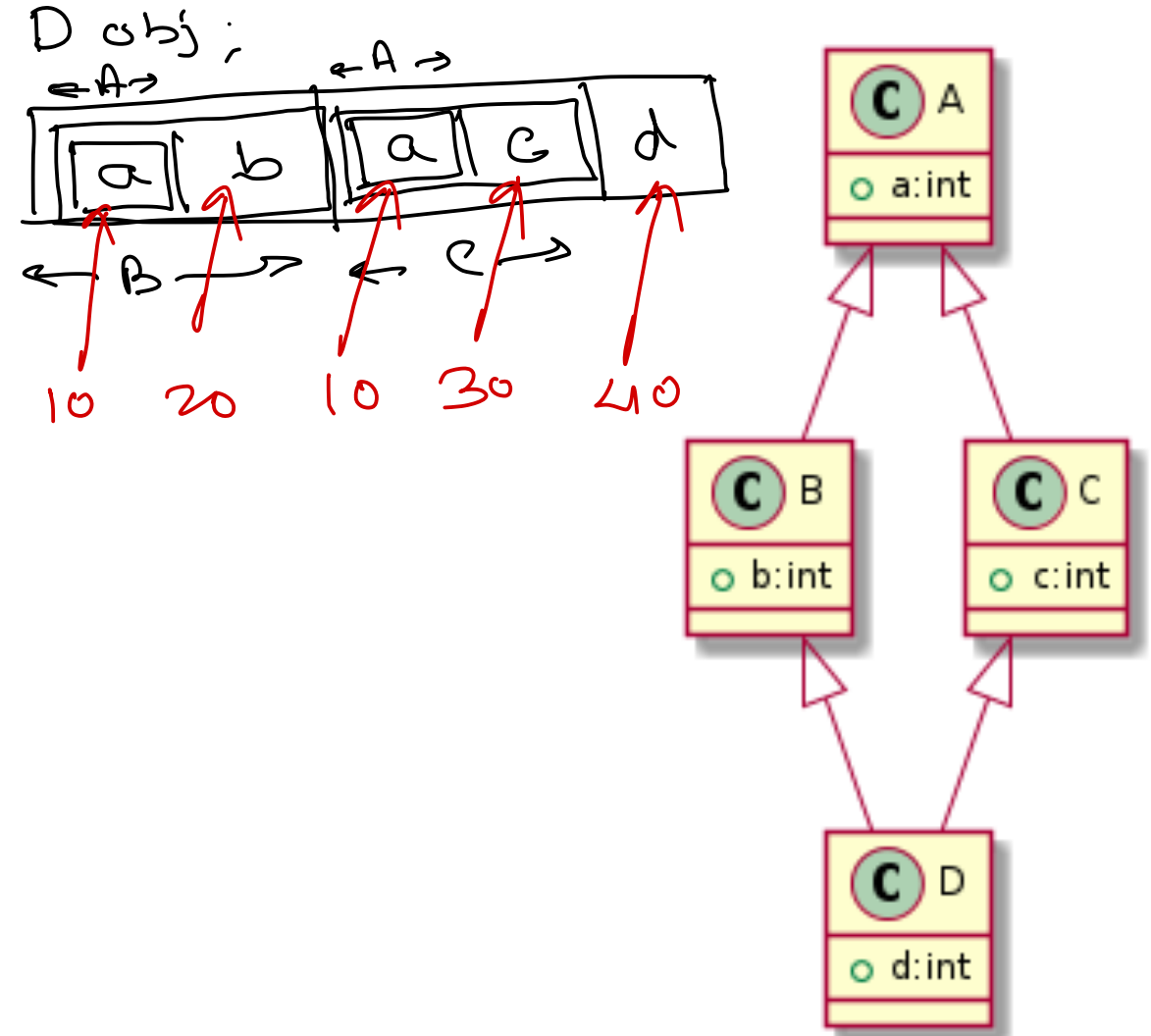- Accessing such members from derived class object will lead to ambiguity error.

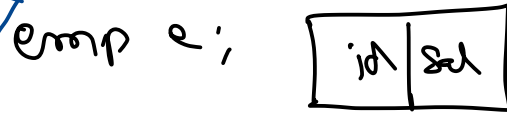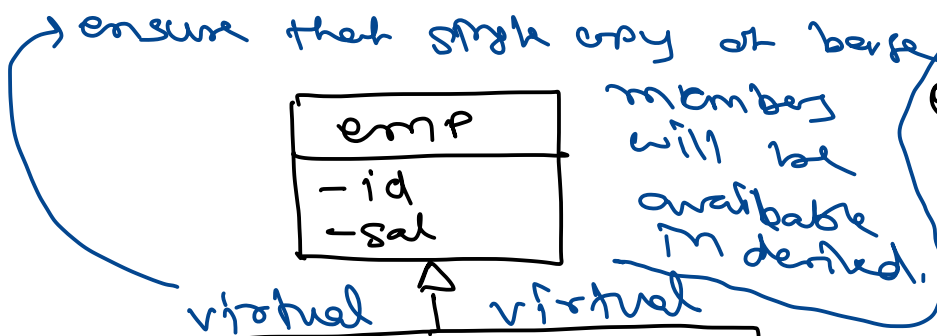- It can be resolved using scope resolution operator.

- sizeof(D) = ?? 20

- Constructor calling sequence

  - A, B, A, C, D.

- Destructor calling sequence will be reverse.
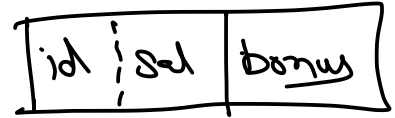
D obj:

←A→		←A→		
a	b	a	c	d
10	20	10	30	40

B →   ← C →

A
a:int

B
b:int

C
c:int

D
d:int

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

emp
- id
- sal

virtual   virtual

manager
- bonus

Salesman
- Comm

Sales_manager

emp e;

id	Sal

manager m;

id	Sal	bonus

Salesman S;

id	Sal	Comm

Sales_manager sm;

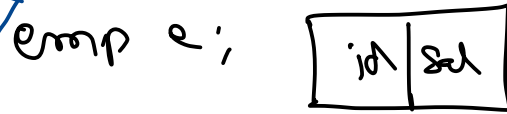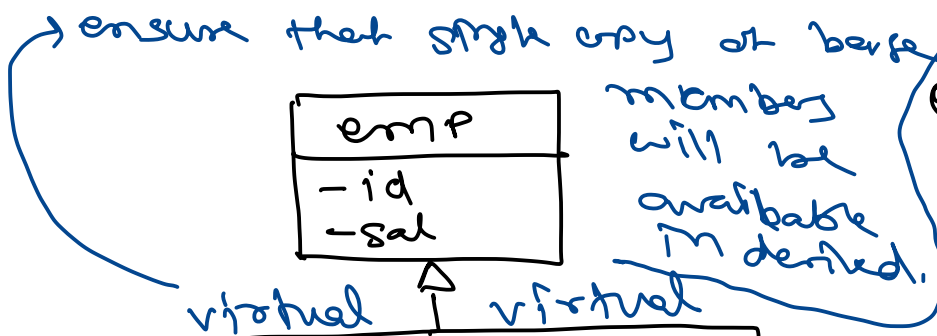| id	Sal	bonus	id	Sal	Comm

← manager → ← Salesman →

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

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

emp e;

id	Sal

emp
- id
- sal

virtual   virtual

manager m;

id	Sal	bonus

manager
- bonus

Salesman
- Comm

Salesman S;

id	Sal	Comm

Sales_manager

Sales_manager sm;

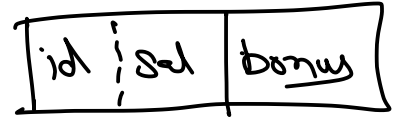| id	Sal	bonus	~~id~~	~~Sal~~	Comm

← manager → ← Salesman →

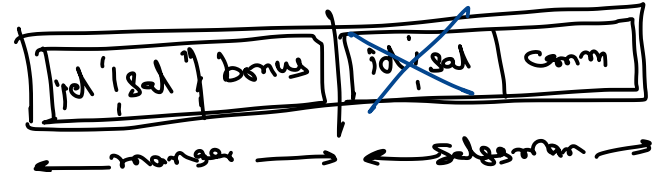In this problem, multiple copies of id & Sal in Sales_manager class are not desirable.

- 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:

vbptr | b | vbptr | c | a | d

8 + 4 + 8 + 4 + 4 + 4

= 32

B obj:

vbptr | b | a

A obj:

a
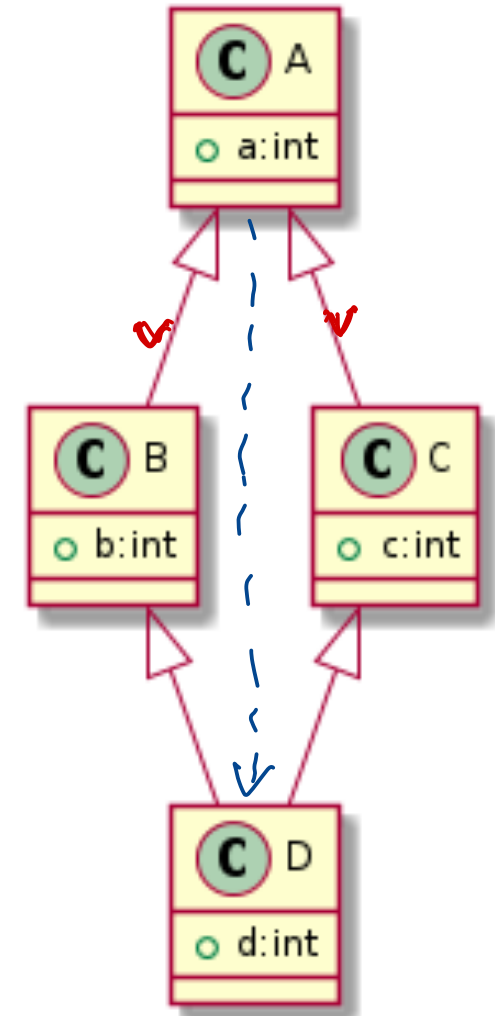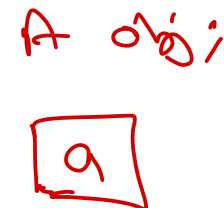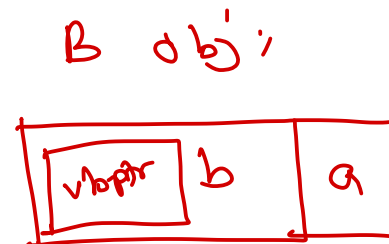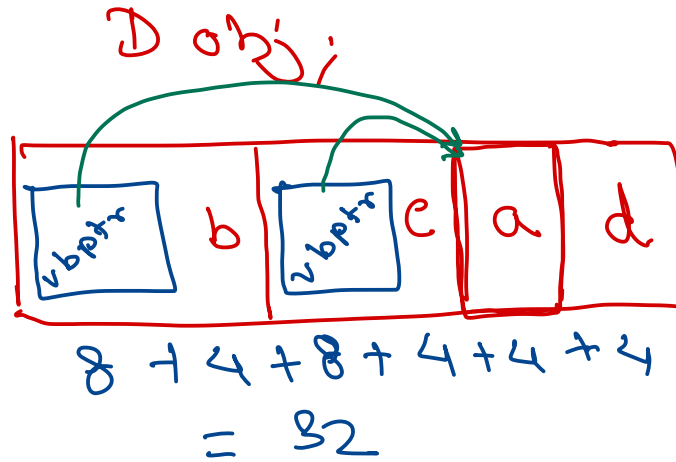
Modes of inheritance

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.)

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();

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

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 → refrence

Method overloading vs Method overriding

runtime polymorphism

- Methods with same name and different arguments. → *for 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 retrun 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).

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".

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>