Dr. D. Y. Patil Pratishthan's
Institute for Advanced Computing and Software Development

Sub-c++
Day6

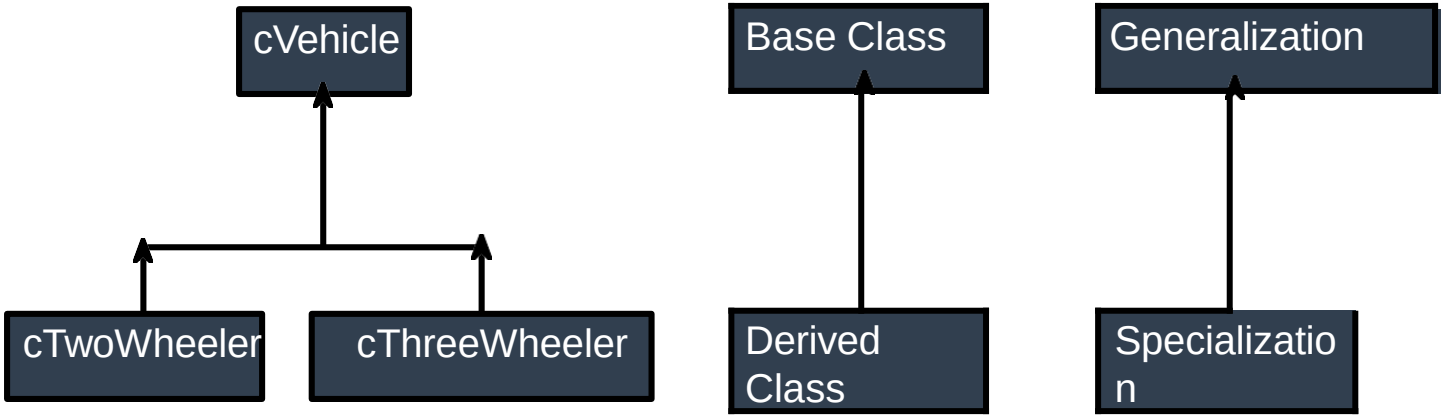
Inheritance

- One of the key-concepts of object-oriented approach.
- Allows creation of hierarchical classification.
- In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.
- In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

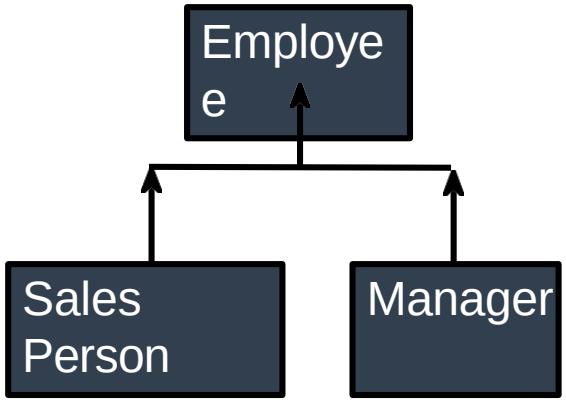
- Advantages of Inheritance
  - Reusability
  - Extensibility

Base and Derived Class

Derived class inherits all data members and methods of its base class.

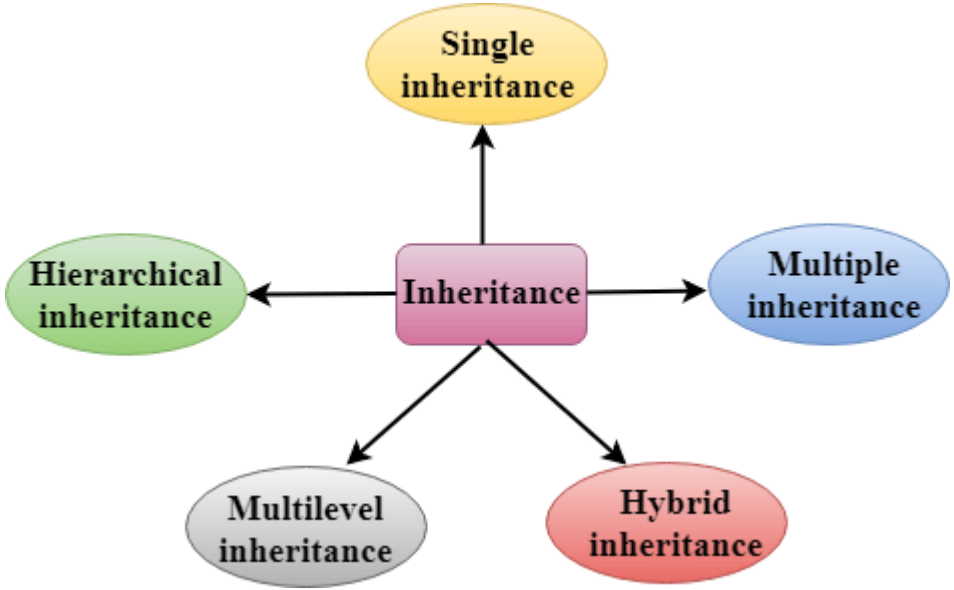


Base and Derived Classes - Example



- This is ‘is a’ kind of hierarchy.
- More than one class can inherit attributes from a single base class.
- A derived class can be a base class to another class.

- **C++ supports five types of inheritance:**
- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance



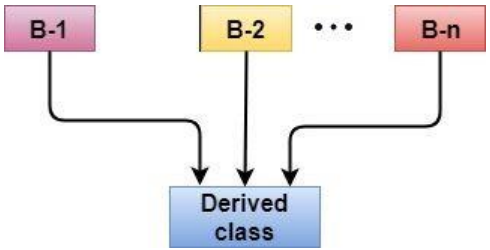
Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

Class B---> Class A

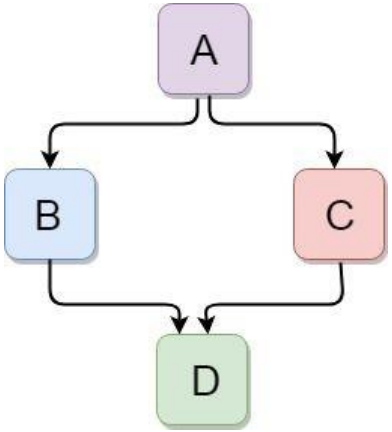
Multilevel inheritance is a process of deriving a class from another derived class.

Class C—>Class B—>Class A

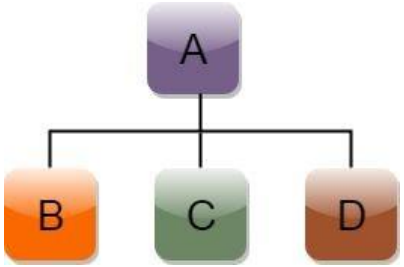
Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.



- Hybrid inheritance is a combination of more than one type of inheritance.



- Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



Inheritance Syntax

- Access / visibilitymode can be **`public`** or **`private`** or **`protected`**.

```
class baseClassName
{
     // body of base class
};
class derivedClassName:visiblity_mode baseClassName
{
    // body of derived class
};
```

Visibility_Modes of Inheritance

- Private
  - Visibility mode: `private`
  - Private members of base class are not accessible in derived class
  - Protected members of base class treated as private in derived class.
  - All public members of base class are treated as private in derived class

- Protected
  - Visibility mode : `protected`
  - Private members of base class are not accessible in derived class
  - Protected members of base class treated as protected in derived class.
  - All public members of base class are treated as protected in derived class

Public

- Visibility mode : public
- All public members of base class become public members of derived class
- All protected members of base class become protected members of derived class
- All private members of base class remain private to base class.

	Derived Class	Derived Class	Derived Class
Base Class	**Private Mode**	**Protected Mode**	**Public Mode**
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Private	Protected	Protected
Public	Private	Protected	Public

Derived Class Constructors/Destructors

- Constructors are called in the sequence of
  - Base -> Derived

- Destructors are called in the sequence of
  - Derived -> Base

Constructors in Derived Classes

- If the base class constructor does not have any arguments, there is no need for any constructor in the derived class
- if there are one or more arguments in the base class constructor, derived class need to pass argument to the base class constructor
- If both base and derived classes have constructors, base class constructor is executed first
- In multiple inheritances, base classes are constructed in the order in which they appear in the class declaration
- In multilevel inheritance, the constructors are executed in the order of inheritance
- C++ supports a special syntax for passing arguments to multiple base classes

- The constructor of the derived class receives all the arguments at once and then will pass the call to the respective base classes
- The body is called after the constructors is finished executing

syntax:

Derived-Constructor (arg1, arg2, arg3….): Base 1-Constructor (arg1,arg2), Base 2-Constructor(arg3,arg4)

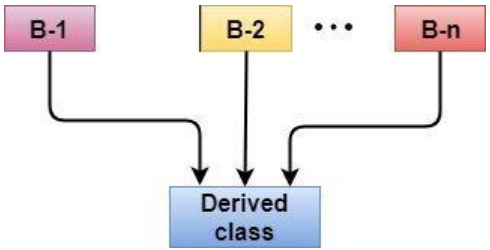
{

….

}

Multiple inheritance syntax

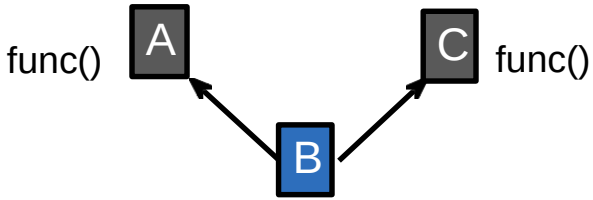
```
class D : visibility  B-1_baseclassname, visibility B-2_baseclassname
{
    // Body of the class;
}
```



Problems of Multiple Inheritance

```
class B : public A, public C { … }
```

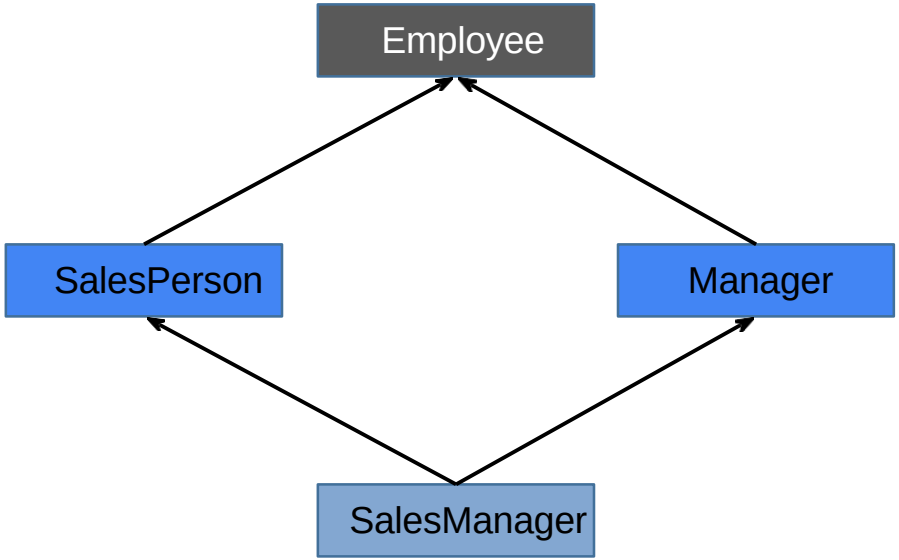
```
B bobj;
bobj.func();
```



1. If multiple base classes contain a function with same name.
   - Resolve by any of the following ways:
     - Use scope resultion operator - `bobj.A::func()` or `bobj.C::func()`
     - Override `func()` in B class.
2. Leads to serious problem of diamond inheritance.
   - Resolve using virtual base class.

Hybrid /Diamond Inheritance: Derive Sales Manager

- When a class inherits from two classes, each of which inherits from a single base class, it leads to a diamond shaped inheritance pattern.



Solution: Virtual Base Class

- Duplicate data member ambiguity can be resolved by declaring a virtual base class.
  - Derive `cSalesPerson` and `cManager` using `virtual` keyword and then derive `cSalesManager` from these two.
- By declaring base class as `virtual`, multiple copies of base class data member are not created.
- There is only one copy of common base class data members in memory and its pointer reference is there in the derived class object.

Virtual Inheritance

- The meaning of `virtual` keyword is overloaded.
- The `virtual` keyword appears in the base lists of the derived classes.

```
class cManager: virtual public cEmployee{
    ……
};
class cSalesPerson: virtual public cEmployee{
    ……
};
```

- The most-derived constructor is responsible for initializing the `virtual` base class.

```
cSalesManager::cSalesManager(…): cEmployee(…),
cSalesPerson(…),cManager(…) {...}
```

Late binding-Virtual Function

- To implement late binding, the function is declared with the keyword `virtual` in the base class.
- Points to note:
  - Virtual function is a member function of a class.
  - Virtual functions can be redefined in the derived class as per the design of the class.
    - Also considered virtual by the compiler

Virtual Function

- It is used to tell the compiler to perform dynamic linkage or late binding on the function.
- There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.
- A 'virtual' is a keyword preceding the normal declaration of a function.
- When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

Virtual Functions

- Some points to note:
  - Should be non-static member function of the base class
  - Generally functions that are overridden in the derived class are declared as virtual functions in the base class.
  - Constructors cannot be declared as `virtual`
  - If a function is declared as `virtual` in the base class then, it will be treated as virtual in the derived class even if the keyword `virtual` is not used.

Method Overloading Vs Overriding

	Overloading	Overriding
Scope	In the same class	In the inherited classes
Purpose	Handy for program design as different method names need not be remembered	Message is same but its implementation needs to be specific to the derived class
Signature of methods	Different for each method overloaded	Has to be same in derived class as in base class
Return Type	Can be same or different as it is not considered	Return type also needs to be same

Pure Virtual Function

- A virtual function without any executable code
- Declared by using a pure specifier `(= 0)` in the declaration of a virtual member function in the class declaration.
- For example, in class `cEmployee`

```
virtual float computeSalary() = 0;
```

- A class containing at least one pure virtual function is termed as abstract class.