



C++ Programming

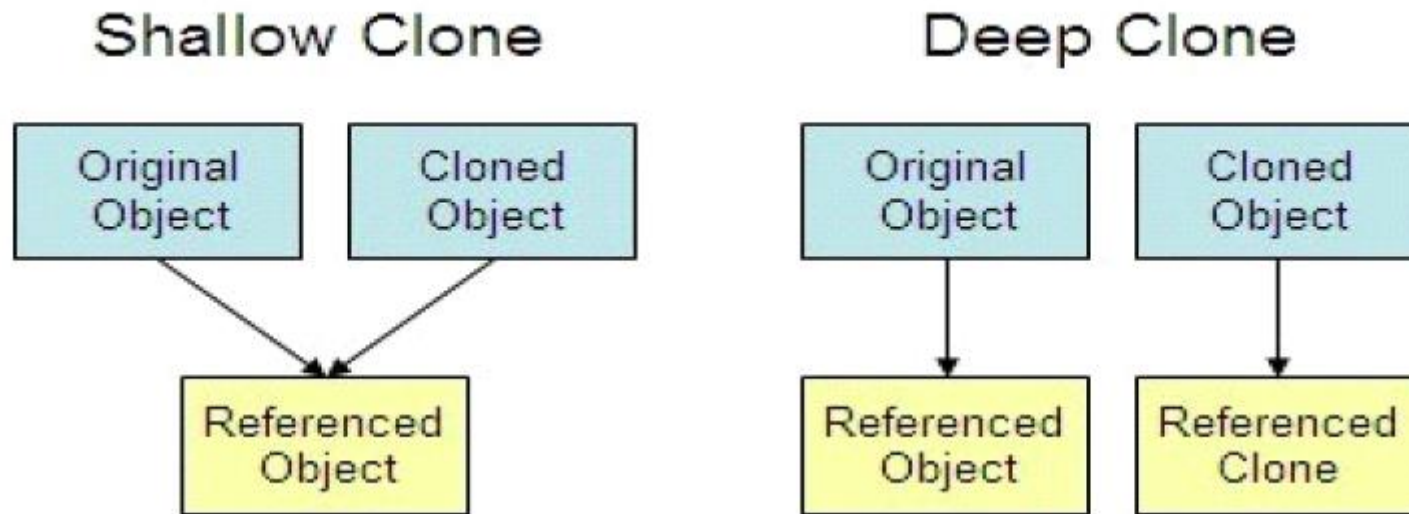
Trainer : Pradnyaa S. Dindorkar

Email: pradnya@sunbeaminfo.com



Object Copying

- In object-oriented programming, “object copying” is a process of creating a copy of an existing object.
- The resulting object is called an object copy or simply copy of the original object.
- Methods of copying:
 - Shallow copy
 - Deep copy



Types of Copy

The copy constructor is used to initialize the new object with the previously created object of the same class.

- **Shallow Copy**

- The process of copying state of object into another object.
- It is also called as **bit-wise** copy.
- When we assign one object to another object at that time copying all the contents from source object to destination object as it is. Such type of copy is called as shallow copy.
- Compiler by default create a shallow copy. Default copy constructor always create shallow copy.

- **Deep Copy**

- Deep copy is the process of copying state of the object by modifying some state.
- It is also called as **member-wise** copy.
- When class contains at least one data member of pointer type, and class contains user defined destructor then when we assign one object to another object then copy constructor is called.
- At that time instead of copy base address allocate a new memory for newly created object and then copy contain from memory of source object into memory of destination object. Such type of copy is called as deep copy.



Object Oriented programming structure(oops) :-

-> It is a programming methodology to organise complex program into simple program in terms of class and objects such methodology is called as "Object Oriented programming structure"

-> It is a programming methodology to organise complex program into simple program by using the concept of Abstraction, Encapsulation and Inheritance, modularity.

->so the language which supports Abstraction, Encapsulation and Inheritance is called as Object Oriented programming language.



Major pillars of oops

- **Abstraction**

- getting only essential things and hiding unnecessary details is called as abstraction.
- Abstraction always describe outer behavior of object.
- In console application when we give call to function in to the main function , it represents the abstraction.
- By Creating object and calling public member function on it we can achieve abstraction.

- **Encapsulation**

- binding of data and code together is called as encapsulation. By defining class we can achieve encapsulation.
- Implementation of abstraction is called encapsulation.
- Encapsulation always describe inner behavior of object
- Function call is abstraction and Function definition is encapsulation.
- Information hiding
 - Hiding information from user is called information hiding.
 - In c++ we used access Specifier to provide information hiding.

- **Modularity**

- Dividing programs into small modules for the purpose of simplicity is called modularity.

- **Hierarchy**

- Hierarchy is ranking or ordering of abstractions.
- Main purpose of hierarchy is to achieve re-usability.
- Types → 1: **Inheritance [is-a]** , 2: **Association [has-a]**



Minor pillars of oops

- **Polymorphism (Typing)**

- One interface having multiple forms is called as polymorphism.
- Polymorphism have two types
 1. **Compile time polymorphism** (Static polymorphism / Static binding / Early binding / Weak typing / False Polymorphism)

when the call to the function resolved at compile time it is called as compile time polymorphism. And it is achieved by using function overloading, operator overloading, template
 2. **Runtime polymorphism** (Dynamic polymorphism / Dynamic binding / Late binding / Strong typing / True polymorphism)

when the call to the function resolved at run time it is called as run time polymorphism. And it is achieved by using function overriding.

- **Concurrency**

- Process of executing multiple tasks simultaneously is called Concurrency.
- Can be achieved by multithreading which is Used to utilize hardware resources efficiently.

- **Persistence**

- Used to maintain state of object across time and space on secondary storage .
- Using file handling we can achieve it. To transfer and save the state of object needs serialization and also socket programming for network.



Association

- If has-a relationship exist between two types then we should use association.
- Example : Car has-a engine (OR engine is part-of car)
- If object is part-of / component of another object then it is called association.
- If we declare object of a class as a data member inside another class then it represents association.

Example Association:

- Car has-a engine
- Laptop has-a hand disk
- Room has-a wall
- Bank has-a accounts

```
class Engine
{
    int cc, fuel;
};
class Car
{
    private:
        Engine e; //Association
};
Dependant Object : Car Object
Dependency Object : Engine Object
```





Example of Association



Composition and aggregation are specialized form of association

Composition

- If dependency object do not exist without Dependent object then it represents composition.
- Composition represents tight coupling.
- Example: Human has-a heart.

```
class Heart
```

```
{ };
```

```
class Human
```

```
{  
    Heart hrt; //Association->Composition
```

```
};
```

Dependent Object : Human Object

Dependency Object : Heart Object

Aggregation

- If dependency object exist without Dependent object then it represents Aggregation.
- Aggregation represents loose coupling.
- Example: Department has-a Faculty.

```
class Faculty
```

```
{ };
```

```
class Department
```

```
{  
    Faculty f; //Association->Aggregation
```

```
};
```

Dependent Object : Department Object

Dependency Object : Faculty Object



Inheritance

- If "is-a" relationship exist between two types then we should use inheritance.
- Inheritance is also called as " journey from Generalization to Specialization".
- Example: Book is-a product
- During inheritance, members of base class inherit into derived class.
- If we create object of derived class then non static data members declared in base class get space inside it.
- Size of object = sum of size of non static data members declared in base class and derived class.
- If we use private/protected/public keyword to control visibility of members of class by using access Specifier.
- If we use private/protected/public keyword to extend the class then it is called mode of inheritance.
- Default mode of inheritance is private.
 - Example: class Employee : person //is treated as class Employee : private Person
- Example: class Employee:public Person
- In all types of mode, private members inherit into derived class but we can not access it inside member function of derived class.



Syntax of inheritance in C++

<pre>class Person //Parent class { }; class Employee : public Person // Child class { };</pre>	<p>In C++ Parent class is called as Base class and child class is called as derived class. To create derived class we should use colon(:) operator. As shown in this code, public is mode of inheritance.</p>
<pre>class Person //Parent class { char name[30]; int age; }; class Employee : public Person //Child class { int empid; float salary; }; int main(void) { Person p; cout<<sizeof(p)<<endl; Employee emp; cout<<sizeof(emp)<<endl; return 0; }</pre>	<p>If we create object of derived class, then all the non- static data member declared in base class & derived class get space inside it i.e. non-static. static data members of base class inherit into the derived class.</p>



Except following functions, including nested class, all the members of base class, inherit into the derived class

- Constructor
- Destructor
- Copy constructor
- Assignment operator
- Friend function.



Protected Data member

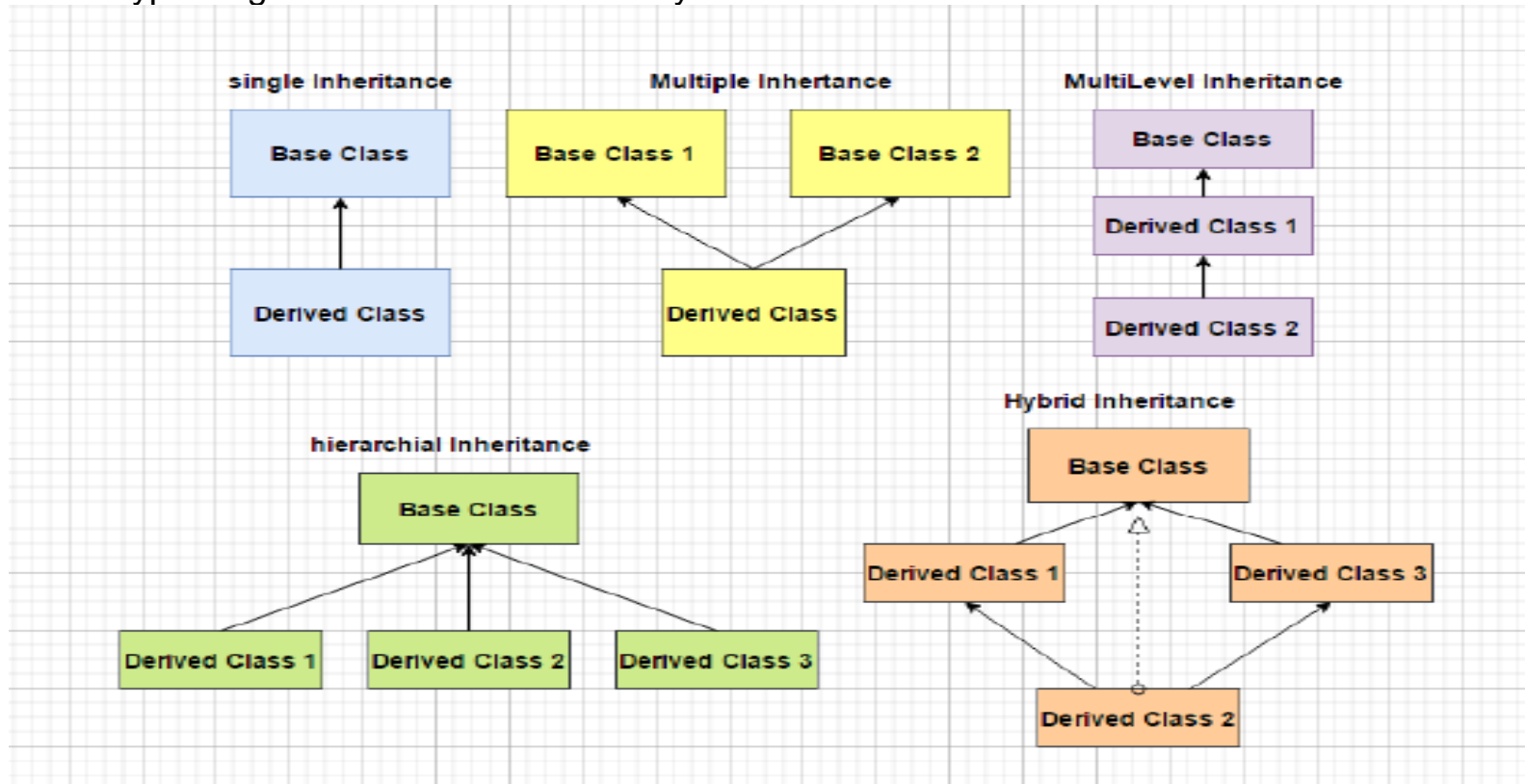
- The protected access specifier allows the base class members to access onto derived class.
- However, protected members are not accessible from outside the class and global functions like `main()`.
- Protected members in a class are similar to private members as they cannot be accessed from outside the class.
- But they can be accessed by derived classes or child classes while private members cannot.



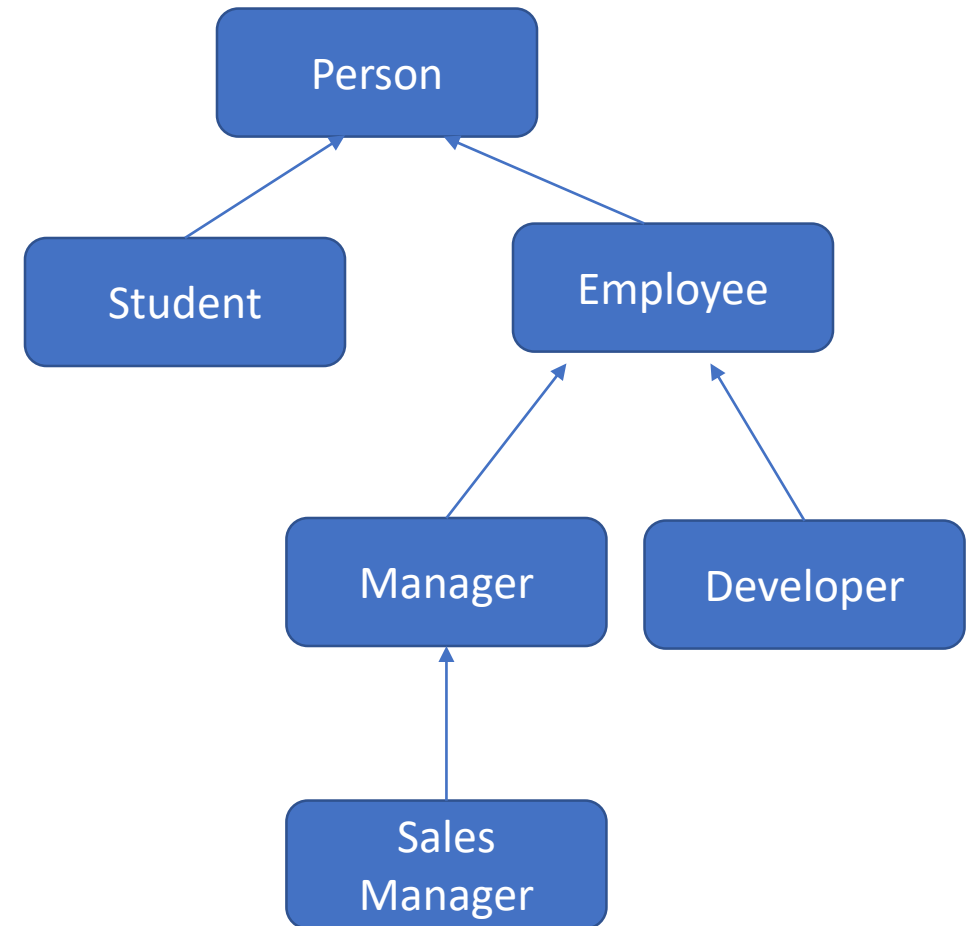
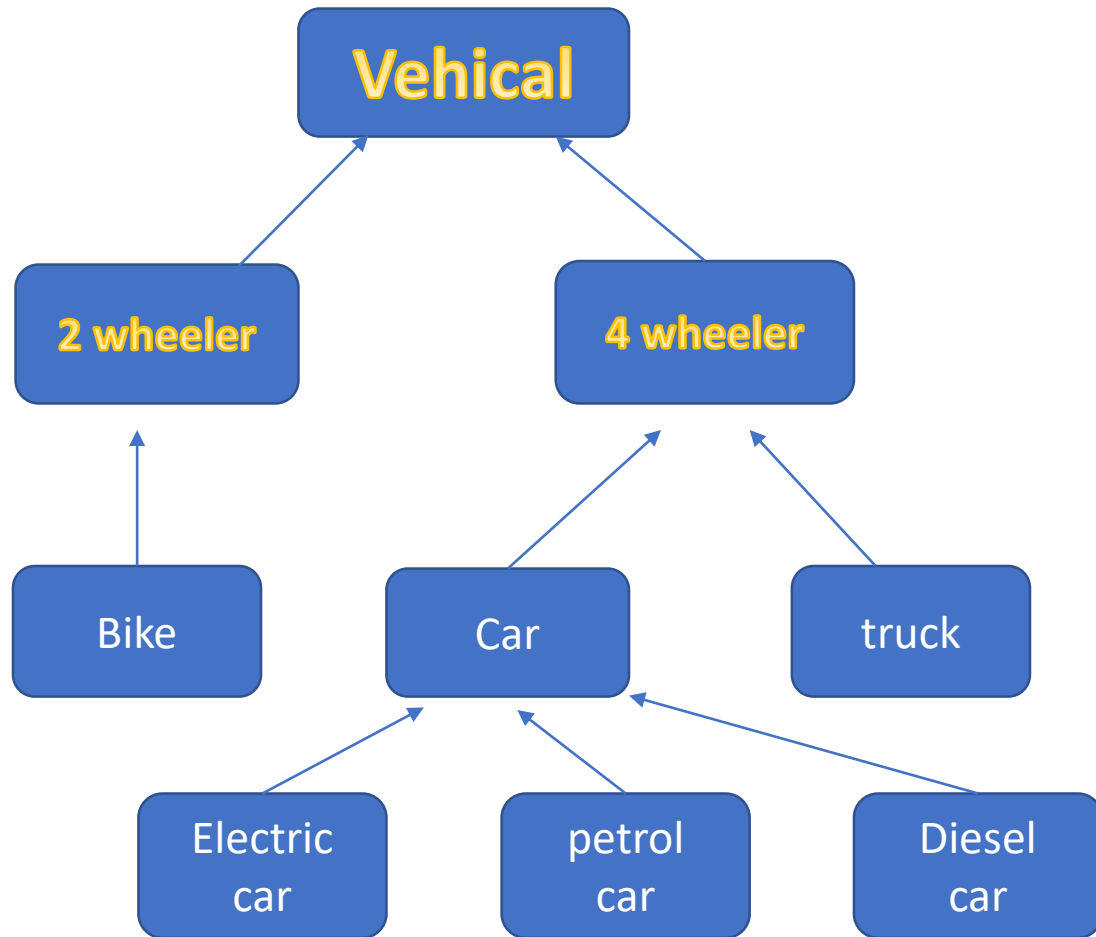
Types of Inheritance

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance

If we combine any two or more types together then it is called as hybrid inheritance.



Inheritance is also called as " journey from Generalization to Specialization".



BOOK

Inheritance

Product

Library

Association

Book

Rice

Inheritance

Food

Bowler

Inheritance

player

Mobile

Association

Charger



Mode of inheritance

- If we use private, protected and public keyword to manage visibility of the members of class then it is called as access specifier.
- But if we use these keywords to extends the class then it is called as mode of inheritance.
- C++ supports private, protected and public mode of inheritance. If we do not specify any mode, then default mode of inheritance is private.



Mode of inheritance

Mode of inheritance (read "--->" as becomes)

Base	Derived
------	---------

public mode:

Public --->	Public
protected --->	Protected
private --->	NA

protected mode:

Public --->	Protected
protected --->	Protected
private --->	NA

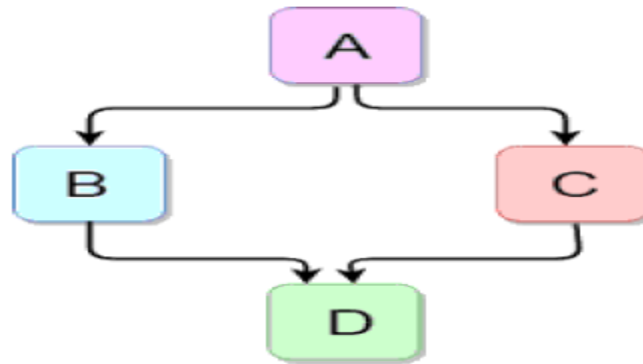
private mode:

Public --->	private
protected --->	private
private --->	NA



Diamond Problem

- As shown in diagram it is hybrid inheritance. Its shape is like diamond hence it is also called as diamond inheritance.
- Data members of indirect base class inherit into the indirect derived class multiple times. Hence it effects on size of object of indirect derived class.
- Member functions 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, then compiler generates ambiguity error.
- If we create object of indirect derived class, then constructor and destructor of indirect base class gets called multiple times.
- All above problems generated by hybrid inheritance is called diamond problem.



Solution to Diamond Problem– Virtual Base Class

- If we want to overcome diamond problem, then we should declare base class virtual i.e. we should derive class B & C from class A virtually. It is called virtual inheritance. In this case, members of class A will be inherited into B & C but it will not be inherited from B & C into class D.

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



Thank You

