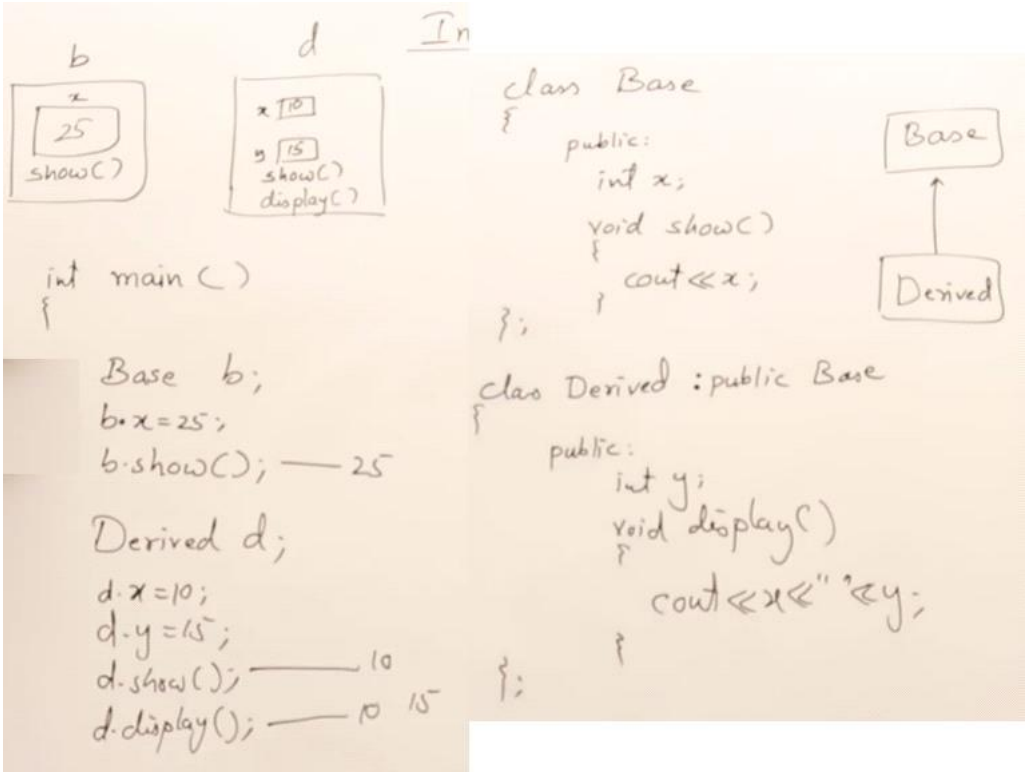
Section 14 : Inheritance

15 April 2022 16:49

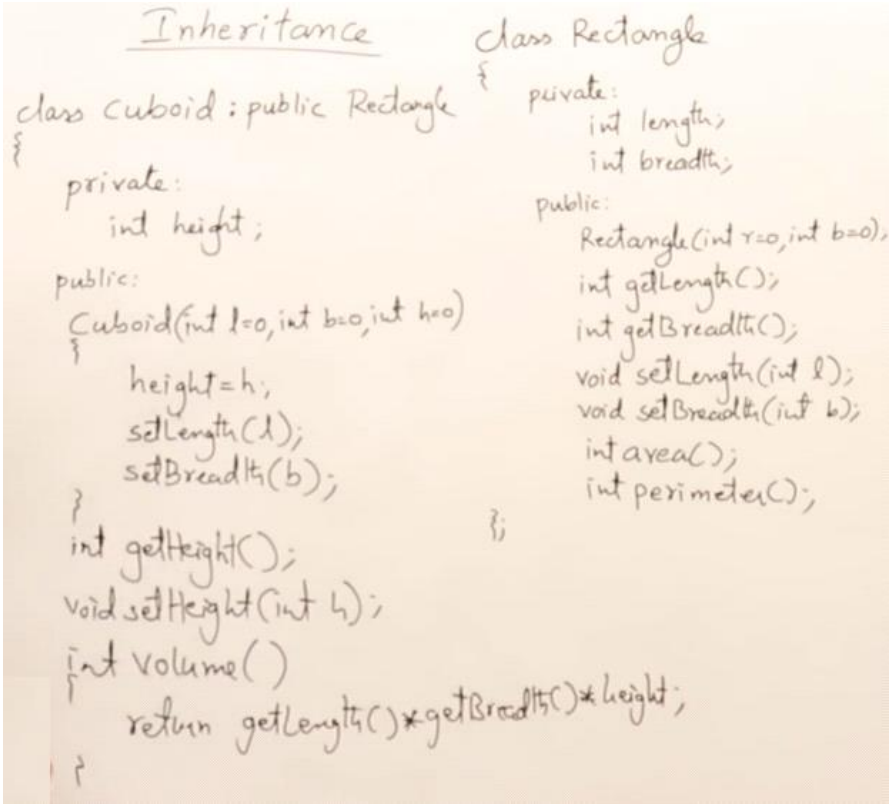
Section 14 : Inheritance

Inheritance :

- Acquiring the features of existing class into a new class, that is deriving class from existing class.
- That is the procedure of borrowing the features of an existing class into a new class of living from this
- Syntax :
    - Class Derived : public Base {…};

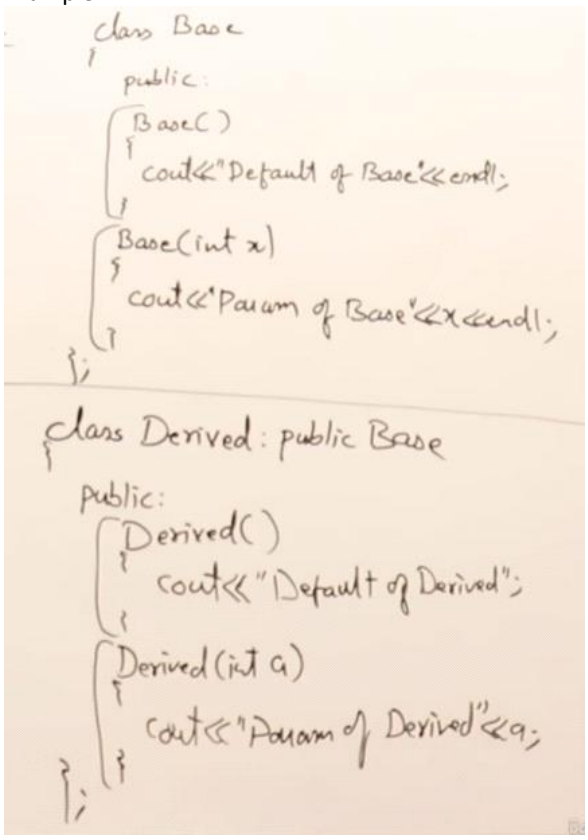


- The private members are not accessible inside the derived class that are declared in base class, so we need to use setters and getters.

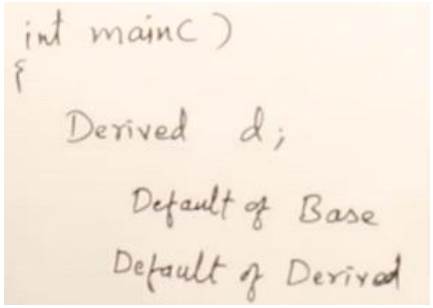


Constructors in Inheritance :

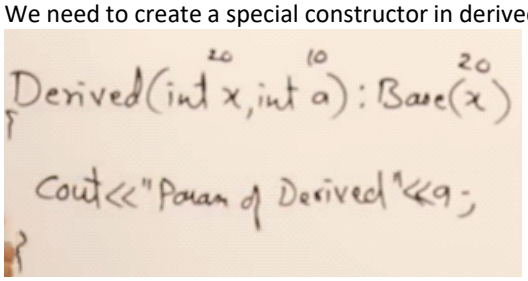
- If we create an object of derived class, then first default constructor of base is called after that derived class constructor will called.
- So it means when we create an object of the derived class, first the base class constructor is executed, then the derived class constructor is executed.
- Example :



- Output :



- Default constructor of base class will executed, doesn't matter parameterized constructor of derived is calling or not.
- What if we want to call parametrized constructor of base class?
  - We can call parametrized constructor of base class from the derived class constructor.
  - We need to create a special constructor in derived class.




```
Derived(int x,int y):Base(x)
{
    cout<<"Param of Derived "<<y<<endl;

}
```

- So the idea is that when we create an object of derive class, first the constructor of base class is executed then the derived constructor is executed.
- So they are called from derived to base but execution is from base to derived.

isA vs hasA :

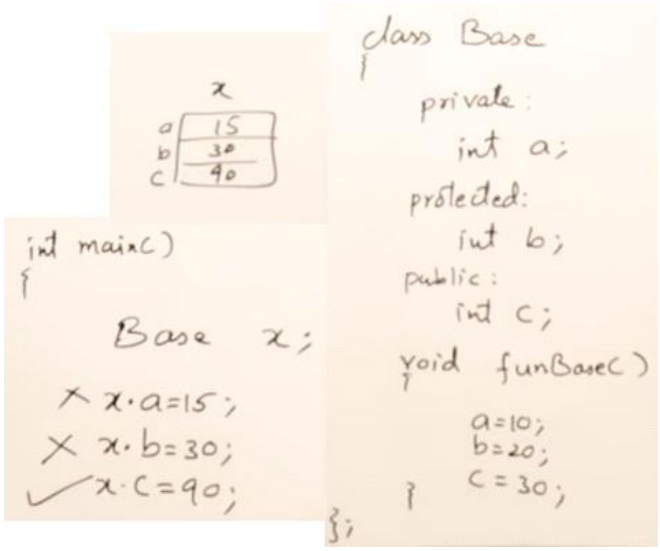
- I have a class called Rectangle and I have a class called Cuboid, which is inheriting from Rectangle, is emerging from Rectangle. So we can say Cuboid is a rectangle.
  - So the relationship between the rectangle class and cuboid class is **isA relationship**.
- Table classes having the table top that is rectangular. So this is having an object of rectangle class table classes. So we can say table class has a rectangle.
  - So the relationship between the table class and rectangle class is **hasA relationship**.

- There are **two ways a class can be used** :
  - One is, a class can be derived, we can write child classes and inherited them from base class.
  - Or the another is objects of that class can be used.
- So there are two ways of using one class, either to create the object and use it or we inherit from it.

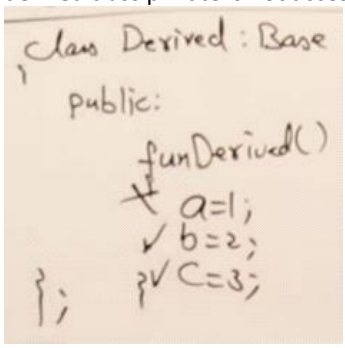
- If a class is inheriting from some class, then it is having **isA relationship** with that one. Or if a class is having an object of some class, then it is having **hasA relationship** with that class.

Access Specifiers :

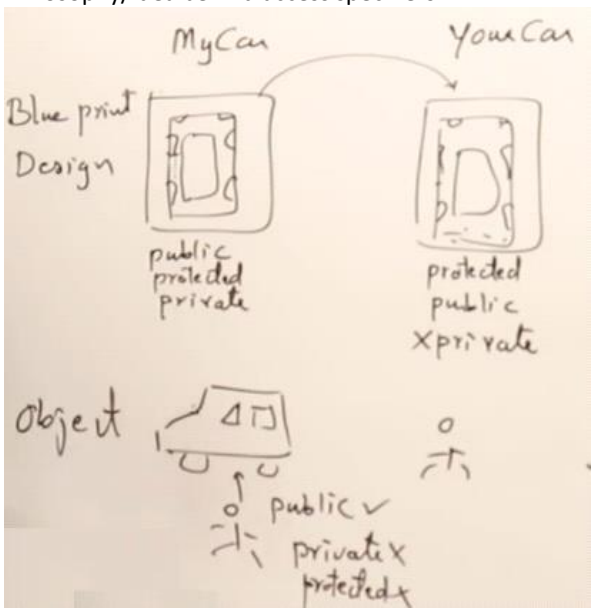
- A class can have three type of members that are
  - Public
  - Private
  - Protected
- When we create an object of a class, we cannot access all members except public members.



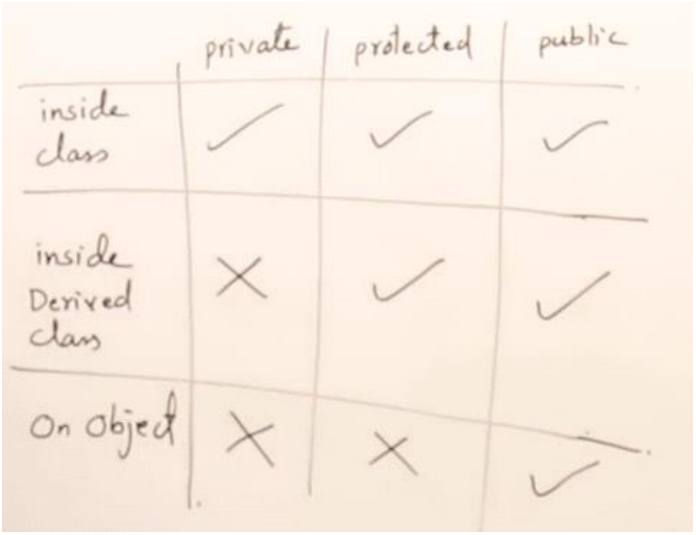
- I have a base class having three members, private, protected and public. All are accessible within this class only, but in the derived class private is not accessible, protected and public are accessible and on an object only public are accessible.



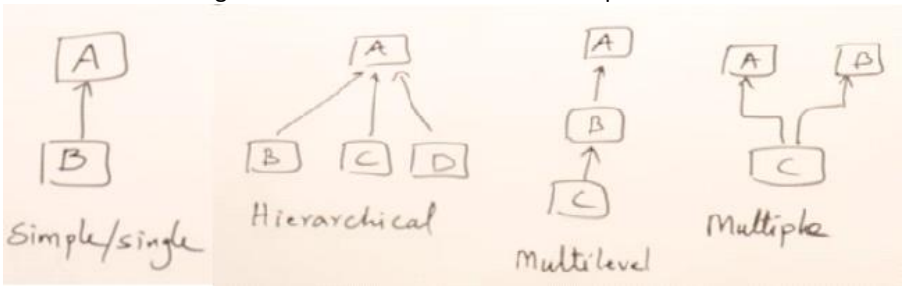
- Philosophy/Idea behind access specifiers



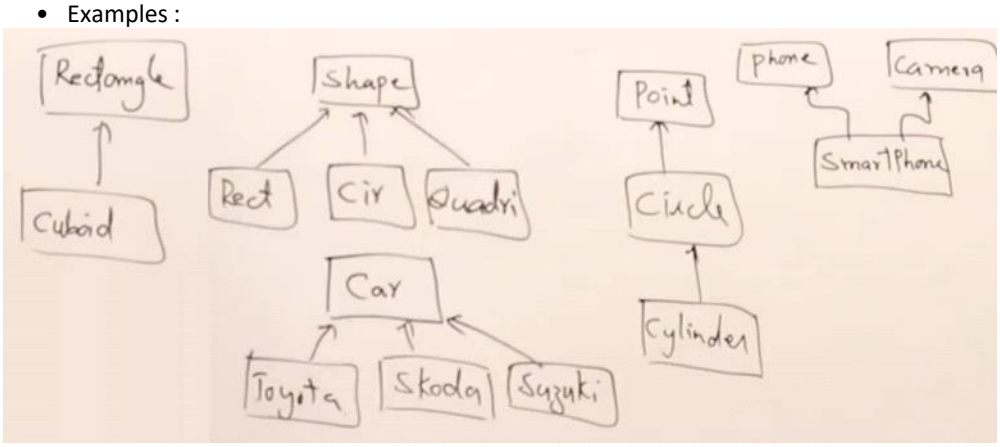
- What are accessible and what are not :



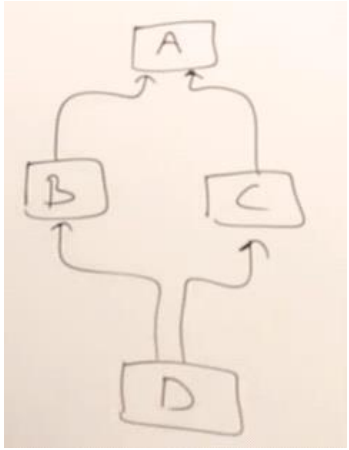
- **Types of Inheritance** :
  - Simple/Single Inheritance
  - Hierarchical Inheritance
  - Multilevel Inheritance
  - Multiple Inheritance - A class can inherit from more than one classes (not possible in java)
  - - mixing hierarchical and multilevel or multiple.



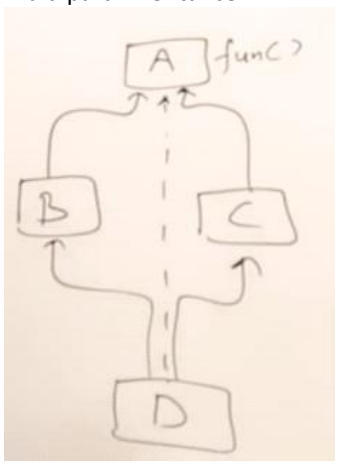
- Examples :



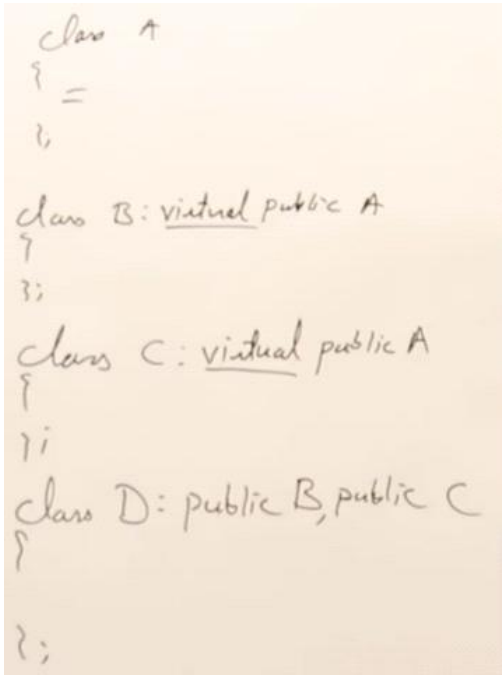
- Hybrid Inheritance : combination of other inheritance



- Multi part inheritance :



  - If there is a function in class A, it will available in class C via 2 paths, by class B or by class C. So there will be ambiguity of that function. To solve this we have the concept of **virtual base classes**.



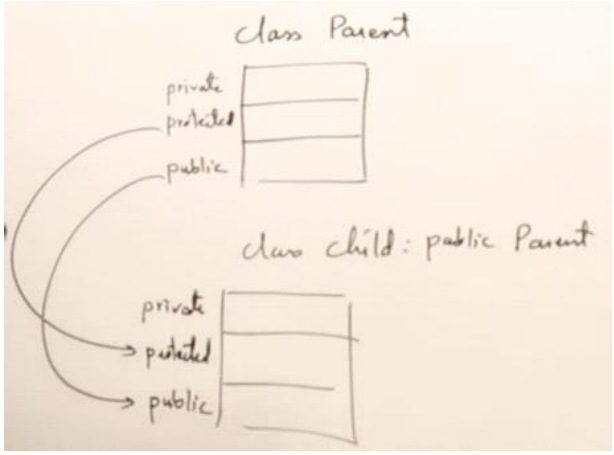
Virtual Base Classes :

- They are useful for removing the ambiguity of the features of parent class in their derived class.

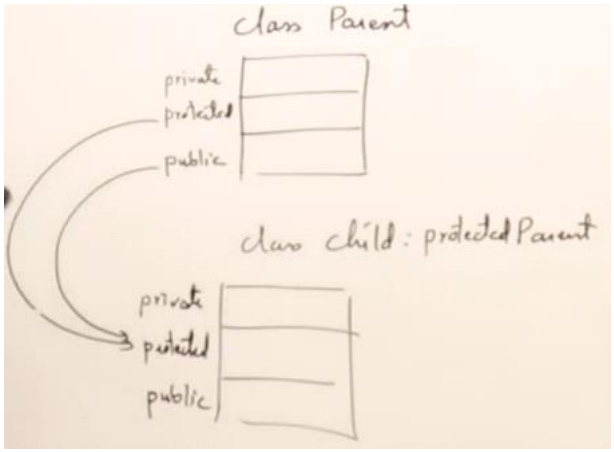
Ways of Inheritance :

- There are more than one way to derive a class from base class.
- There are three methods of inheritance that is publicly, privately and protectively.
- If we are not writing anything, then by default it become privately..

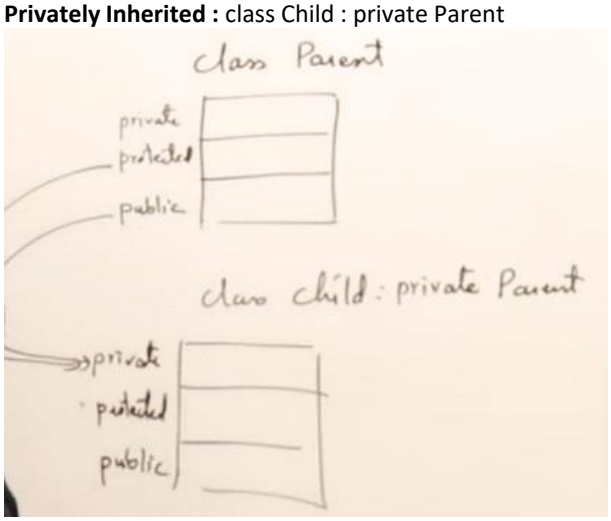
- **Publicly Inherited** : class Child : public Parent



- **Protectively Inherited :** class Child : protected Parent



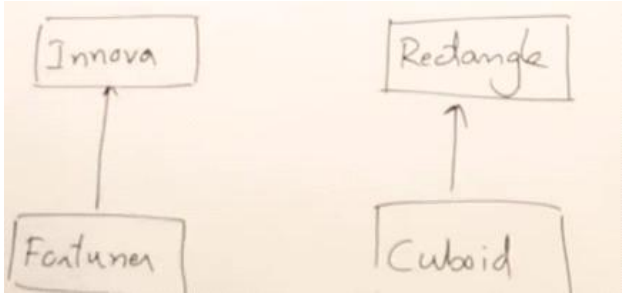
- **Privately Inherited :** class Child : private Parent



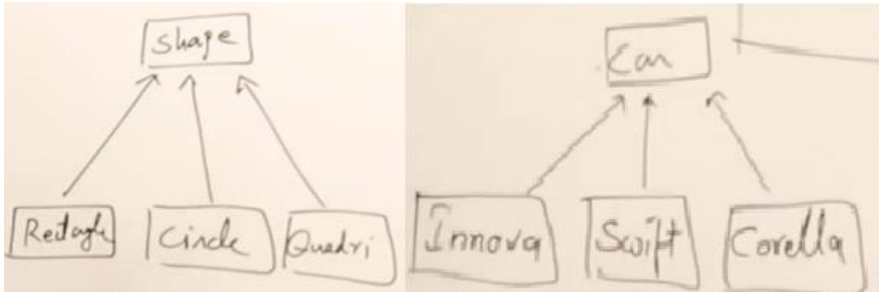
- The method of inheriting will affect the objects and grandchild classes. The derive class has no effect
- In c++, we can restrict access down the hierarchy of classes. (It's not available in other languages, only one type of inheritance in java)

Generalization vs Specialization :

- We have a class rectangle and a class cuboid.
- The rectangle was existing from there we have derived cuboid. So rectangle class is already existing and we have defined a new class with extra features. So we have a **specialized** class that is cuboid.
- Something is already existing. Then we are deriving something from that one and defining a new class. This is **specialization** from real world.
- The child process were existing. Then we define a base class. So this is a top down approach, that **specialization**.
- Here are the base class have something to give it to child class.
- The purpose of **specialization** is to share its features to it's child class.



- We have a rectangle, circle, and Quadratic classes. All derived from shape class. So here shape is a general term, we can't draw/show a shape only, it must be something like rectangle, circle, square, or anything. Shape is logical term, not real not physical. It means share is a **generalize** term. This is an example of generalization.
- Like car is also a general term. Innova, swift, Ferrari we can say they are a car.
- So that is easy for communication in real world, daily life, we define such general terms.
- This is a bottom up approach. This is a **generalization**.
- Here the base classes doesn't have anything to give to the child classes.
- Their purpose is to group them together.



- So we using same word here, It's a car. So this is nothing but polymorphism, same name, but different objects or different actions or different things. That is nothing but **polymorphism**.
- The purpose of **generalization** is to achieve **polymorphism**.

So the purpose of **inheritance** are :

- Share its features to child classes.
- Second is to achieve polymorphism.

- Constructors can be declared as private.