Types of Inheritance: (Modes)

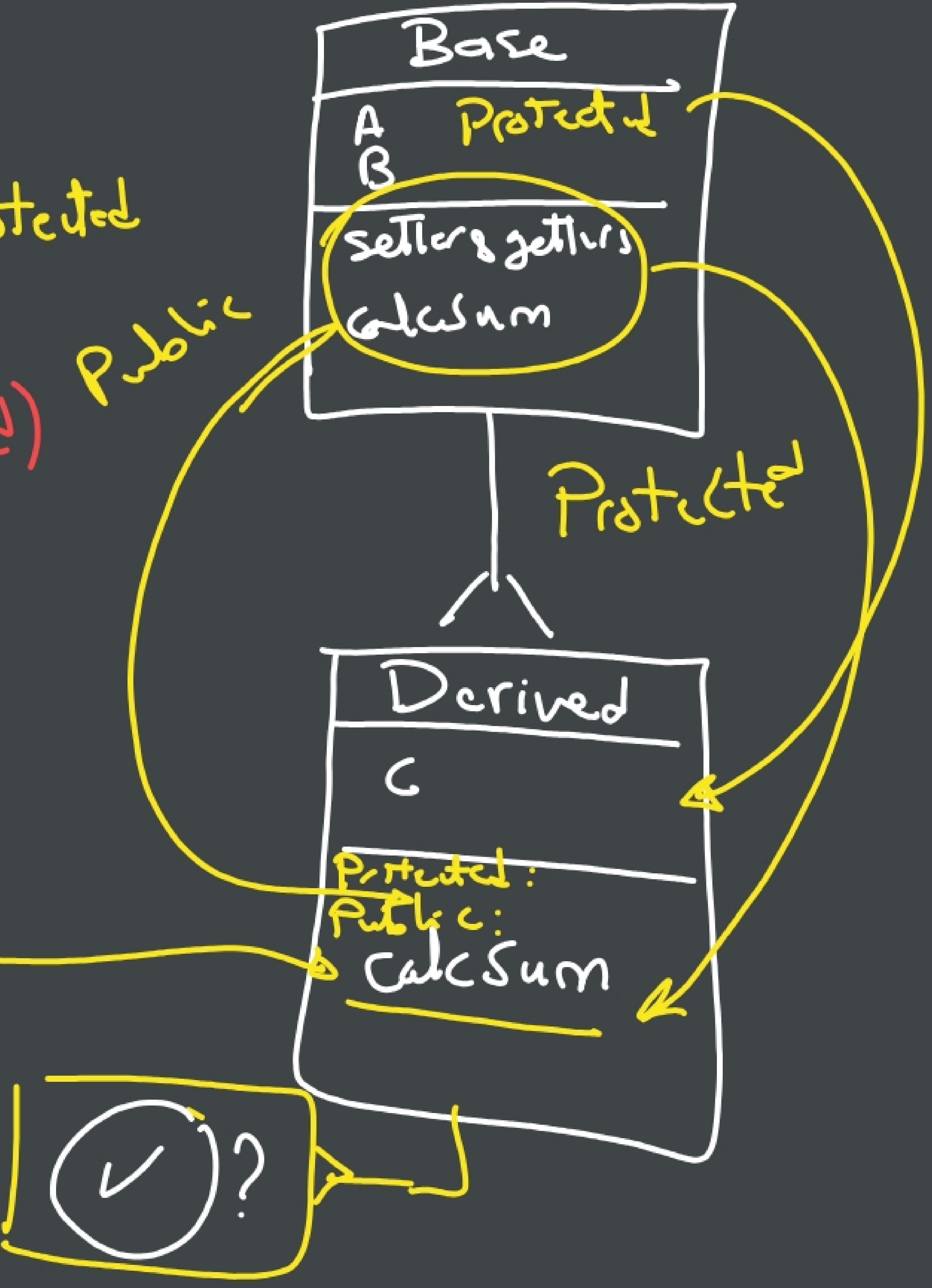
1) Public Inheritance.
Derived inherit all data & methods with same Level of Accessibility in Base class.

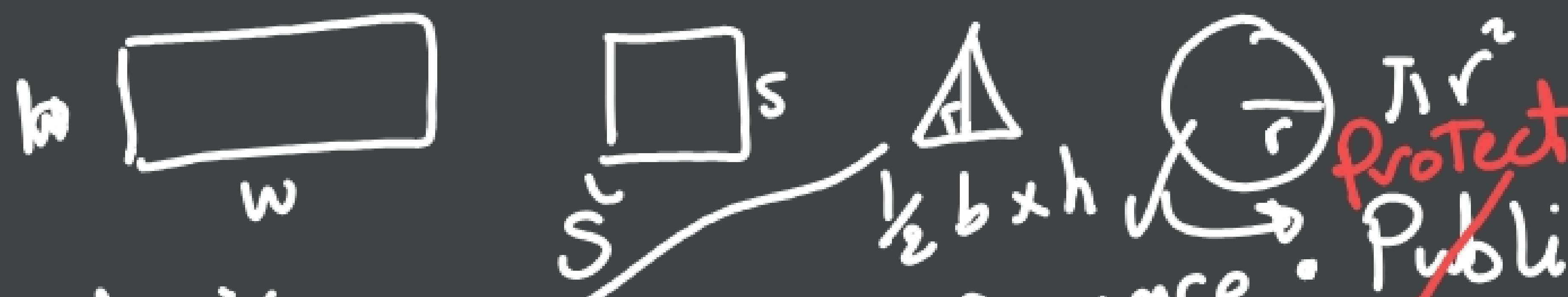
2) Protected Inheritance
Derived inherit (take all methods & Data as there were Protected in its Class.

3) Private Inheritance.
inherit all Data & method as if (Private in Derived)

Derived d;
d.

main()
{
Derived D;
D. calcSum
setA() → X






```
class Geoshape
{ Protected:   4    6
     float dim1, dim2;        Public
  Public:                     inheritance
    Geoshape ( ) = default;
    Geoshape (float d): dim1(d), dim2(d){}
    Geoshape (float d1, float d2) : dim1(d1), dim2(d2){}
    float calcArea () { return 0.0;}
    void PrintDims (){ ______ };
    float getdim1 ( ) { return dim1;}
    float getdim2( ) { return dim2;}
    void setdim1 (float d){ dim1 = d;}
    void setdim2 (float d ){dim2 = d;}
};
class Rect : Public Geoshape          Rect( ) = default;
{ Public:  // using Geoshape::Geoshape;  //inherit all
    Rect (float d1, float d2): Geoshape(d1, d2){ }.
    Float CalcArea( ) { return dim1 x dim2;}
};
```

compiler call Par(ameter) of Base

if not will use call default of Par(ameter).

```
Class Square : Protected ~~Public~~ Rect
{ Public:
    Square ( ) = default;
    Square (float s): Rect (s, s) { }
    { }
    float calcArea ( ) { dim1 x dim2;}
    void setside (float d ) { dim1 = dim2 = s;}
};

main ( )
{ Rect r1 (4, 6);
  r1 . CalcArea( );
  r1 . PrintDimensions( );
  ~~Rect r~~;  r1.SetDim1 (7);
               r1.setDim2(8);
  r1. CalcArea( );

  Square SS(10);
  SS. CalcArea( );  X
  SS.setdim1(7);
  SS. Setdim2(8);
  SS. CalcArea( );
```

call default Rect

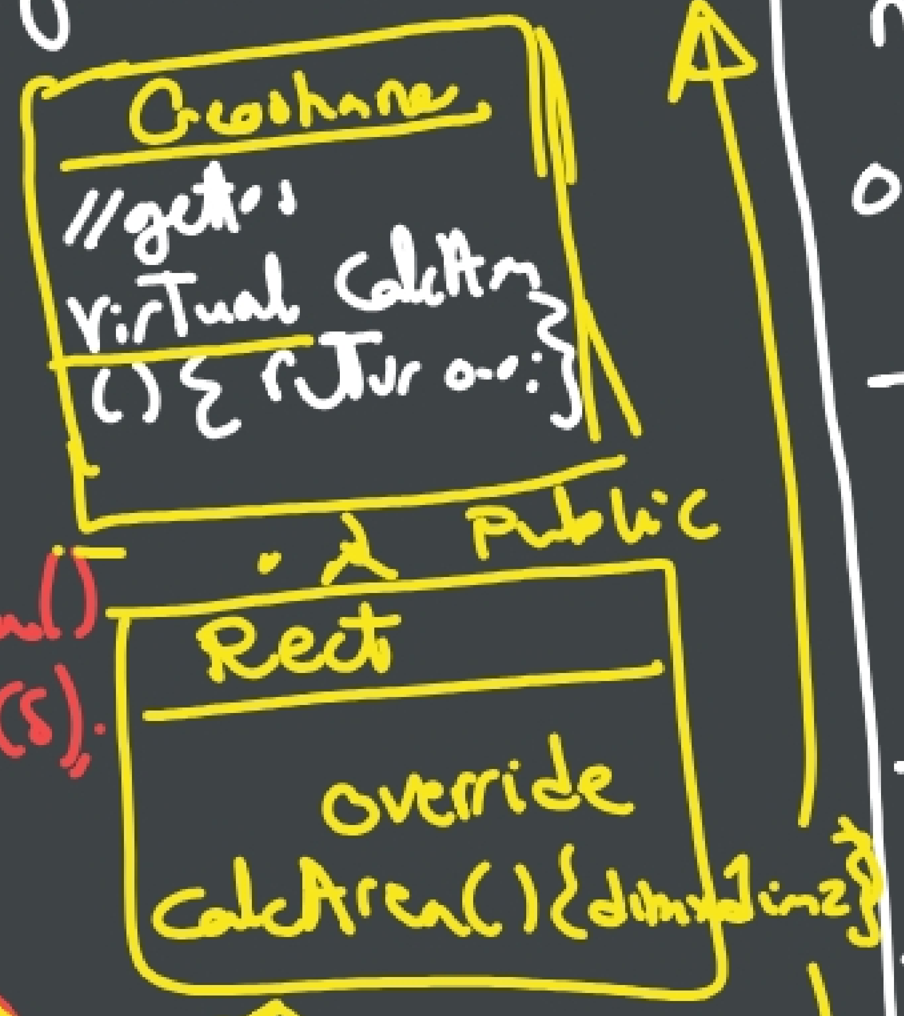
call Parameter of Rect

Dynamic Binding & Virtual function:

Rule 1: A Pointer of a Base class Can Point to the addres of an object of Derived class.

```
main()
{ Geoshape *P;
   Rect myR(3,5);
   Circle myC(10);
   P = &myR;
   cout<< P->calcArea();      15.0
   P = &myC;
   cout<< P->calcArea();
}
```

myR.calcArea()
myR.setdim(5).



- By writing virtual, it will look from down To up To get nearest object function,
- if not writting virtual, it will go up To Pointer method. (Base class fn)

Virtual fn: Key word written in Base class & automatically it applies To rest of derived classes.

- it allow us which method to call during runtime not during compile time because it doesn't know which object will call it actually. (address of Pointer @ call Time).
- virtual can only be applied if type of inheritance Public any other type (mode) of inheritance will give Error.
- Derived classes should override virtual method To implent its own logic.

Rule 2: A Pointer of Base Class Can call overriden method of derived class if the original method in Base was declared as "Virtual".

```
Float calcArea() override { ___ }
```

Derived class

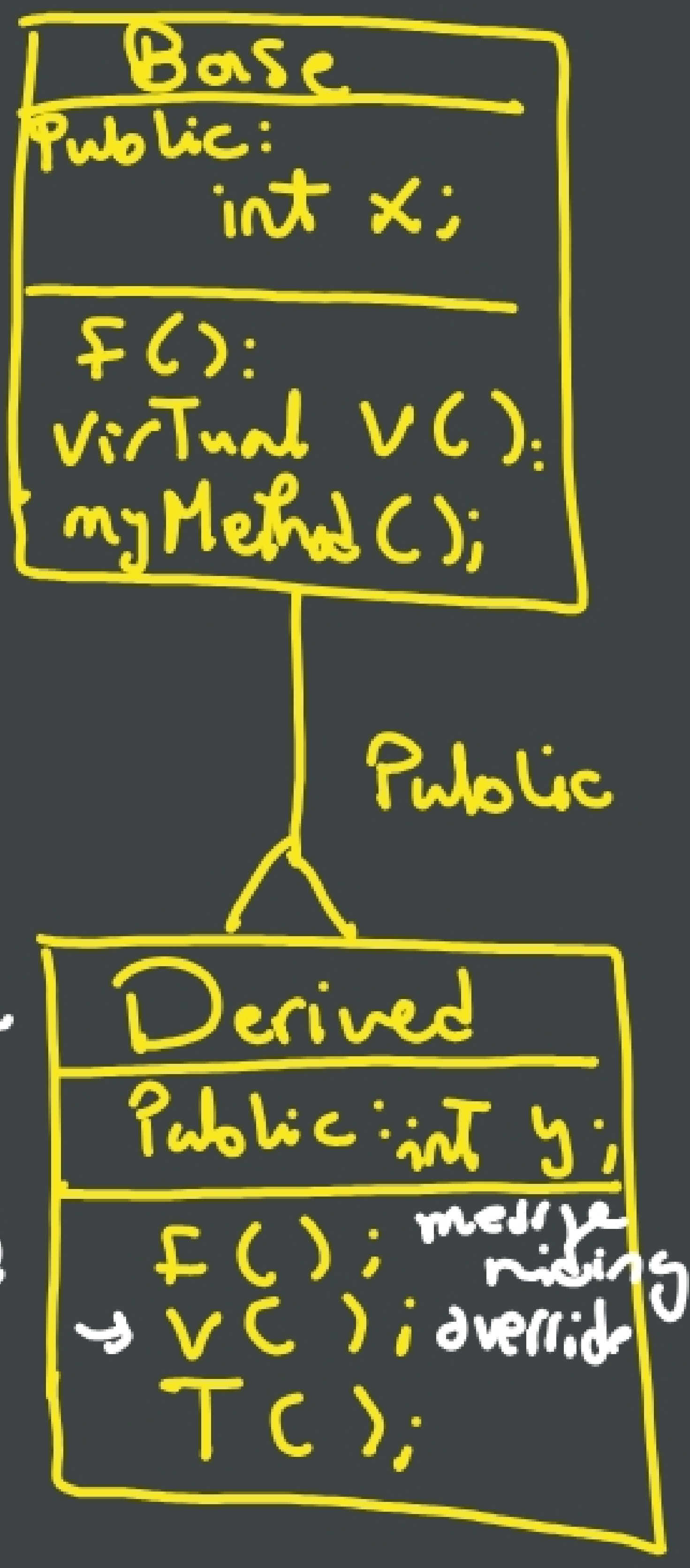
```
// Standalone
float SumAreas (Geoshape *p1, Geoshape *p2, Geoshape *p3)
{   return   p1->calcArea() + p2->calcArea() +
             p3->calcArea();
}

int main ()
{     Rect R(5,7);
      Square S(7);
      Circle C(10);
      Triangle T(5,20);
  Cout<< SumAreas( &S, &T
  Cout<< SumAreas(&R, &C
```

Rule 3:

A Pointer of a Base class Can access only The variables & methods That were originally declared in Base class & Then inherited & Overriden in Derived class.

```
int main()
{   Base * P;          ✓
    Derived obj;       ✓
    P = & obj;         ✓
    P->x = 3;          ✓
    P->y = 7;          ✗ Can't access
    P->F();            — Call F of Base
                         not virtual
    P->V();            → Call V Derived
    P->T();            ✗ Can't access
    P->myMethod();     ✓
                         Base
}
```



→ Turn Geoshape To be Abstract class (Template to use its fund & override it if needed).

```
class Geoshape
{  Protected:

   Public:

   virtual float calcArea() = 0;  ← Pure virtual Function.
   //setters & getters ✓
};
main()
{   Geoshape shape; // Compilation Error.
```

one pure virtual inside Class → became AbsTract Class.

```
Class Rect : Public Geoshape
{

   // must overrid Pure virtual.
   float calcArea() override { return dim1*dim2; }

};
```

Abstract class:

1- Can't create object from it, Just a Pointer.
2- Any derived Class That doesn't override Pure virtuals → it Turned to be Abstract too.
3- You Just Put at least one Pure Virtual inside class → became AbsTract.

Diff between Concrete class & AbsTract class

- Create objects from it Statically or Dynamic

- Could contain Virtual functions To be overriden by its children (Derived classes) → should override not must override

- Just Pointer of Type Abstract Class, Can't create any Type of objects.

- any class inherit from it <u>musT</u> override all its Pure Virtual function, or it will inherite <u>Abstraction</u>.

→ Multible Inheritance

? Selfstudy

Thank You..

Assignments

- Geoshape Example.

  Design rest of classes Circle, Triangle (Applying features of changing Types of inheritance).

- implement Print function for all shapes That show each shape area & Perimeter → Public inheritance what should I do? → Protected inheritance what happened or need to be changed in Design.

- Add Rhombus, Cube → Shapes (who's Parent class).

  - which type of inheritance I should use.

- implement Area, Volume.
- create stand alone function That compare any 2 shapes 2gether → Area
- in Previous Example ⇒ Consider creating CTrs in each Class That is suitable To Class design → show when. To use default of Base & when create my own.
- overLoad = operator in each Class of shapes.

- Turn Geoshap into Abstract class
  & change what must be changed in your design.
- Create main function → use objects Dynamically created.
  → Try creating Pointer of Base.
  - after creating & calculate areas, volume, Perimter
    of shapes → create sort to Print
    all shapes ASC, Desc.