Object Oriented Programming

C → procedural oriented

- Security
- Reusable

OOP

What is OOPs

OOPs is a programming technique which mainly revolve around real life objects

- Data Type

(i) primitive
(ii) Non - primitive
(iii) User defined

int x = 'y' (char

int arr()

Classes & Objects

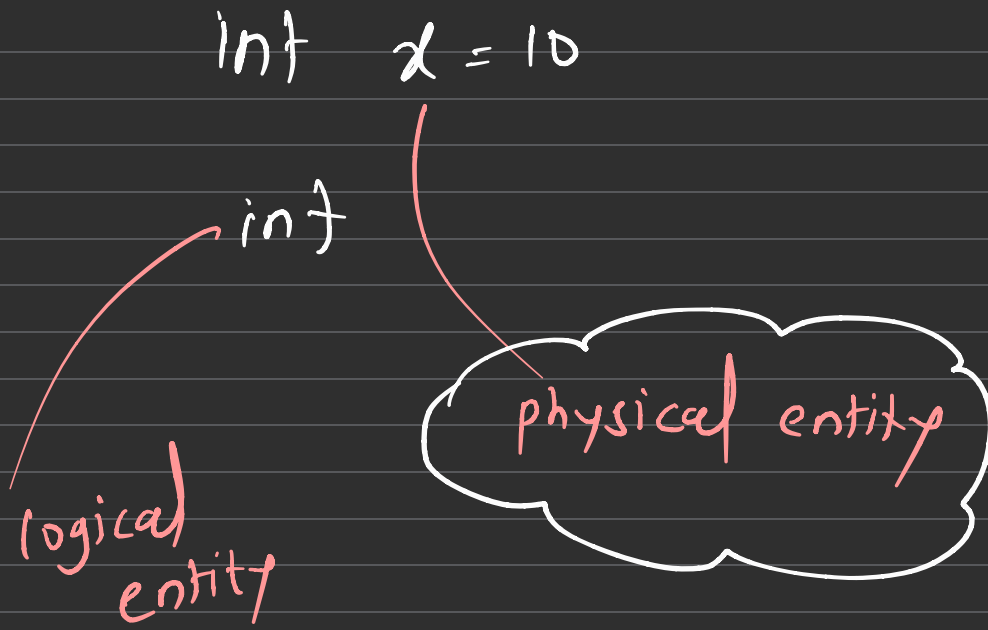
Class

It is user defined data type, which contains data members (variables) & methods (functions).

Syntax:

```
class ClassName
{
    // data member

    // methods
}
```

```
class Person
{
    int age
    sting nm

    void show()
    {
        print (nm, age)
    }
}
```



Object

It is a instance of the class - which tells what kind of operation we can perform on the class.

CPP

Syntax:

```
className  obj

Person  pramod

        int  x
```

How to access members of the class

```
class Person
{
    int age
    string nm

    void show()
    {
        print (nm, age)
    }
}
```

Person obj

(.) dot operator

obj.age

obj.nm

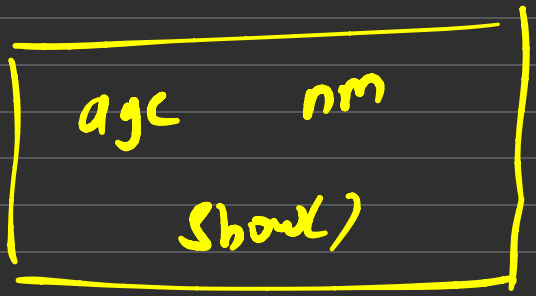
Obj.show()

int x = 10

x

cout << x ⟶ 10

Obj



cout << obj

Access Specifiers

Mainly decides the scope of accessibility of members of the class

~~default~~

private (by default)

public

Protected

inheritance

Private.

- By default all members are private
- can be accessed only in class.

Public

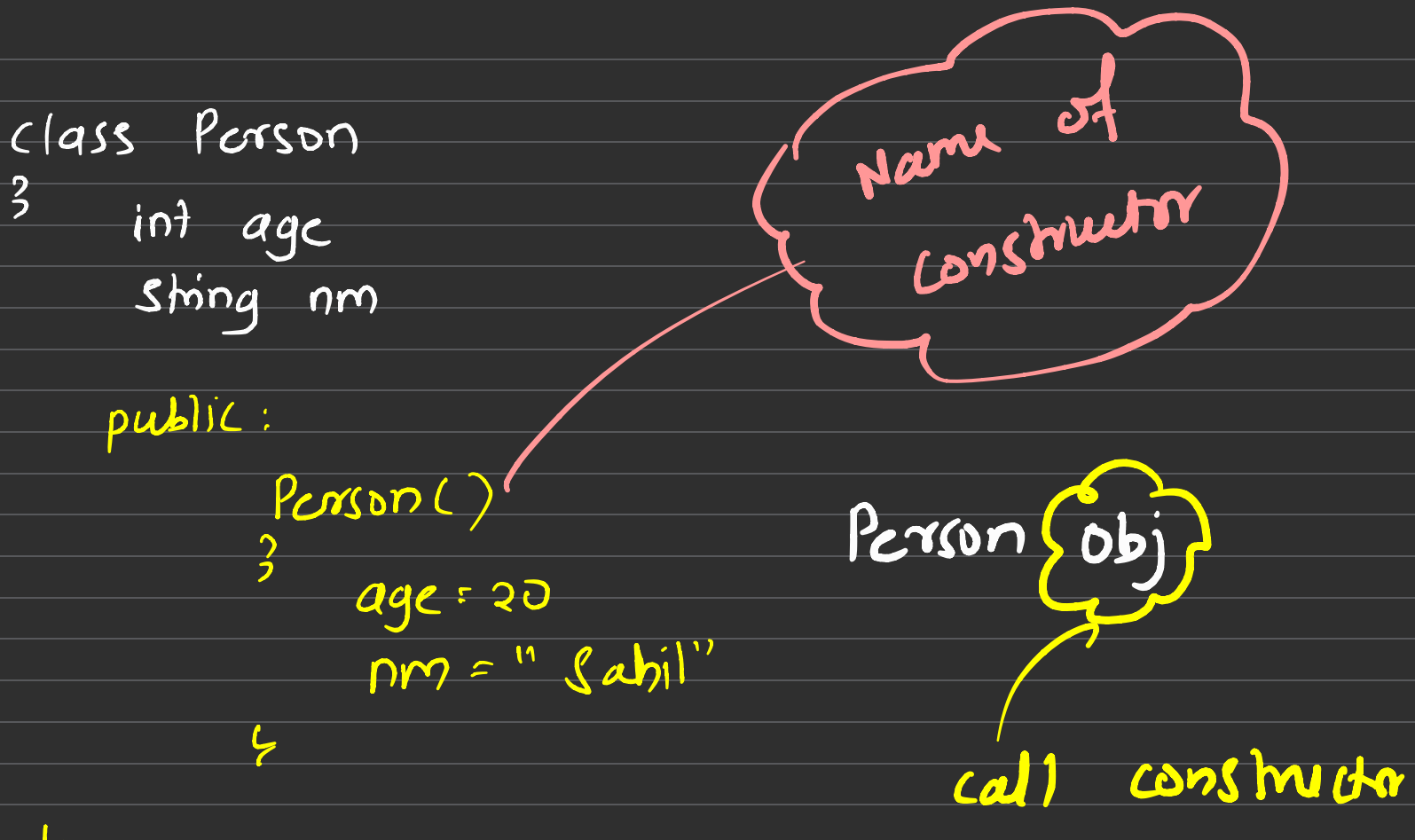
- Anyone can accessed (inside / outside) class

Implict → compiler Explict → program

Constructor & this pointer

mainly helps to initialize object of class

1. always public
2. Name of constructor = name of class
3. does not have return type → not even void
4. calls automatically → moment you create your object



- if no constructor is defined → then behind the Scene default Constructor Exist

- The moment you declar your own constructor → default no longer Exist

1. Default
2. Parameterized
3. Copy constructor

This pointer

```
class Person{

private:
    int age;
    string name;
    int marks;


public:

    // paratemized constructor
    Person(int age, string name, int marks){

        age = age;
        name = name;
        marks = marks;
    }
```

- mainly used to initialize current obj

- also used to separate local variable & class variable

Person (int age, String name, int marks)

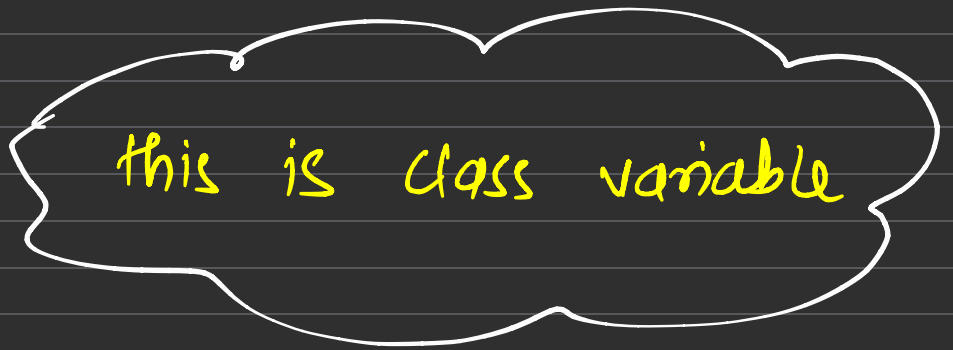
{

this → age = age

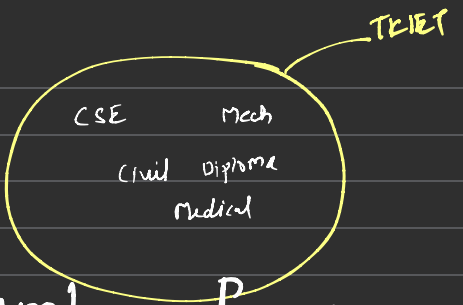
this → name = name

this → marks = marks

}



Encapsulation



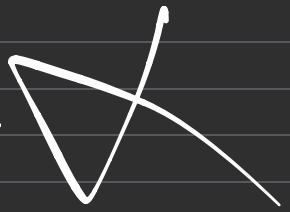
Binding of data member and number functions which performs operation on data members together in same class

```
class Person
{
    int age, marks

    void show()
    {

    }
};
```

- Mainly helps us for data Hiding

```
class Person
{
  private
      int age
      string nm
public
      void show()
      {
          print()
      }
}
```

```
class Area
{
  private
      int radius
      int dia
public:
      void findArea(int height)
      {
          dia
          height = height x100
      }
};
```



485. Max Consecutive Ones

Solve

Easy | Topics | Companies | Hint | Re-do

Given a binary array `nums`, return *the maximum number of consecutive* `1`*'s in the array*.

Example 1:

```
Input: nums = [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are consecutive 1s. The maximum number of consecutive 1s is 3.
```

Example 2:

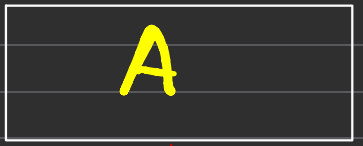
```
Input: nums = [1,0,1,1,0,1]
Output: 2
```

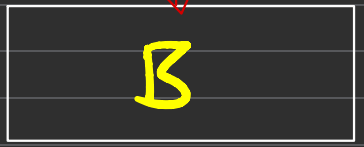
Inheritance

to inherit properties of another class

The capability of a class to derive properties and characteristics from another class is called Inheritance.

parent
Base
super





child
Derived
Sub-class

Syntax Inheritance

CPP

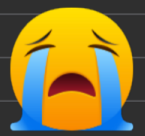
```cpp
// Base Class
class Person{


};

// Derived Class
class Pramod: public Person{



};
```

JAVA

```java
// Base Class
class Person{  3 usages  1 inheritor

}

// Derived Class
class Pramod extends Person{  2

}
```

```cpp
// Base Class
class A{


};

// Base class
class B{


}

// Derived Class
class C: public A, public B{



};
```

In java, we cannot exdend more than one class



Mode of Inheritance

Public mode: If we derive a sub class from a public base class. Then the public member of the base class will becom public in the derived class and protected members of the base class will become protected in derived class.

Protected mode: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.

Private mode: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

```
class Person
{
    private
        int age
        string name

};
class Pramod : public Person
{
        int height

};
```

protected
~~private~~

Base

private > protected

private > public

protected > public

Derived

private > protected > pub

age

name

Types of Inheritance

Single Inheritance

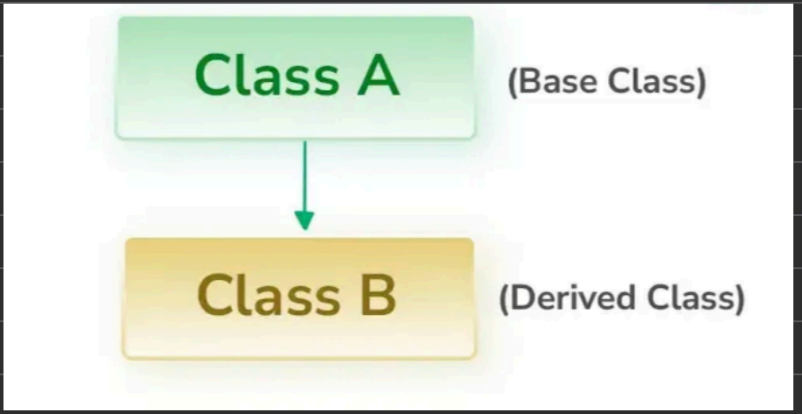
Multilevel Inheritance

Multiple Inheritance ✗

Hierarchical Inheritance

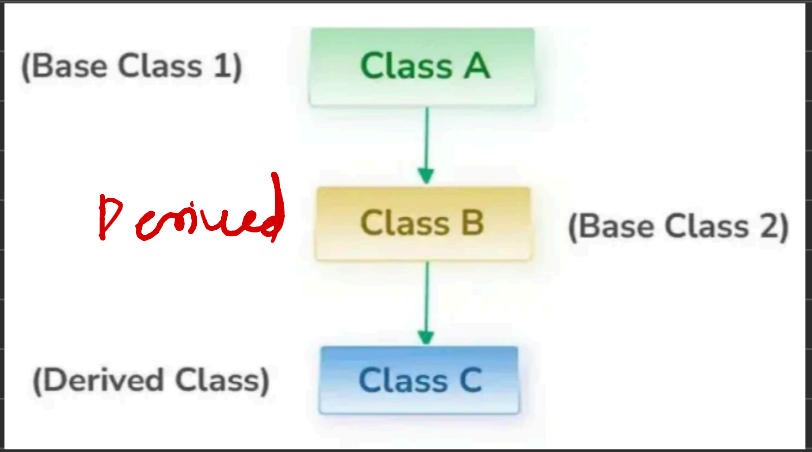
Single Inheritance

In single inheritance, a class is allowed to inherit from only one class. i.e. one base class is inherited by one derived class only.



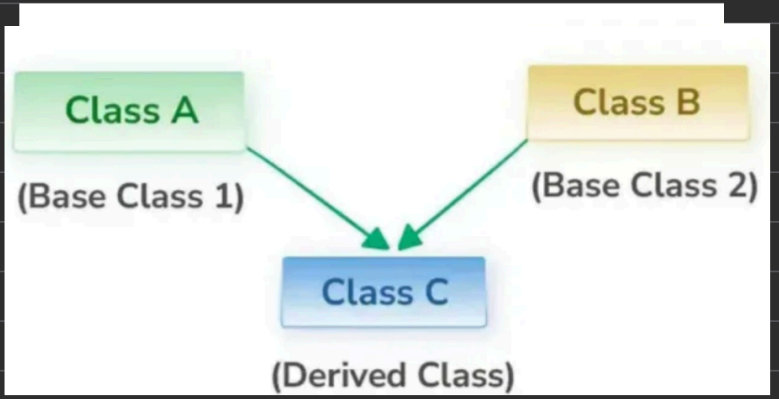
Multilevel Inheritance

A derived class is created from another derived class and that derived class can be derived from a base class or any other derived class. There can be any number of levels.



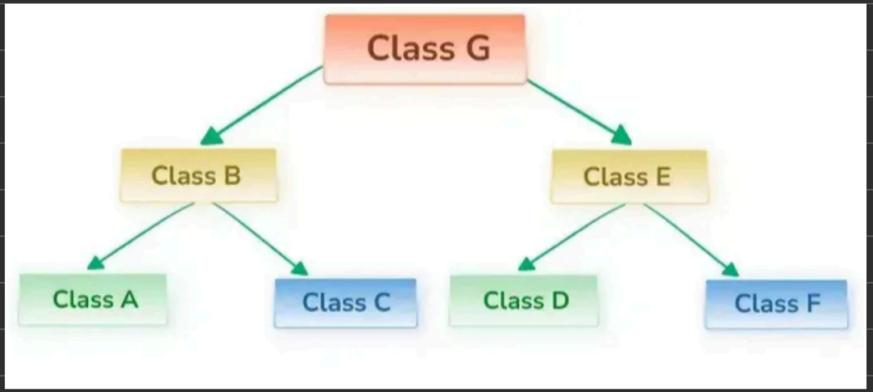
Multiple Inheritance

Multiple Inheritance where a class can inherit from more than one class. i.e one Derived is inherited from more than one Base Class.



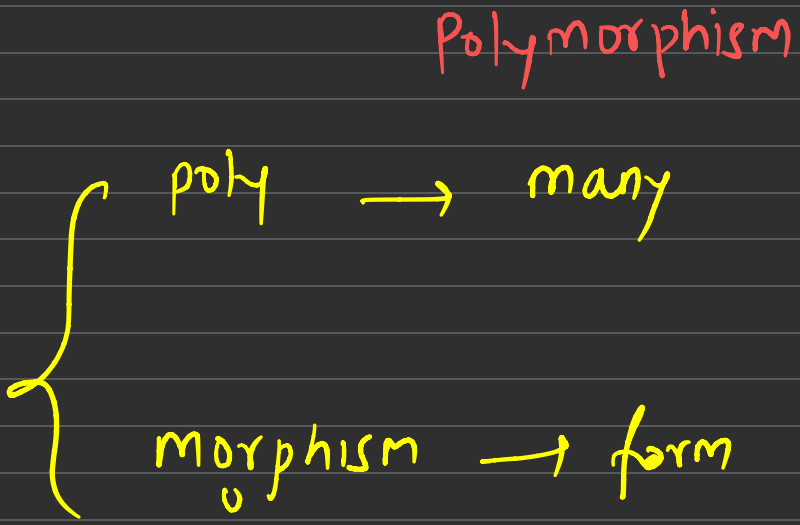
Hierarchical Inheritance

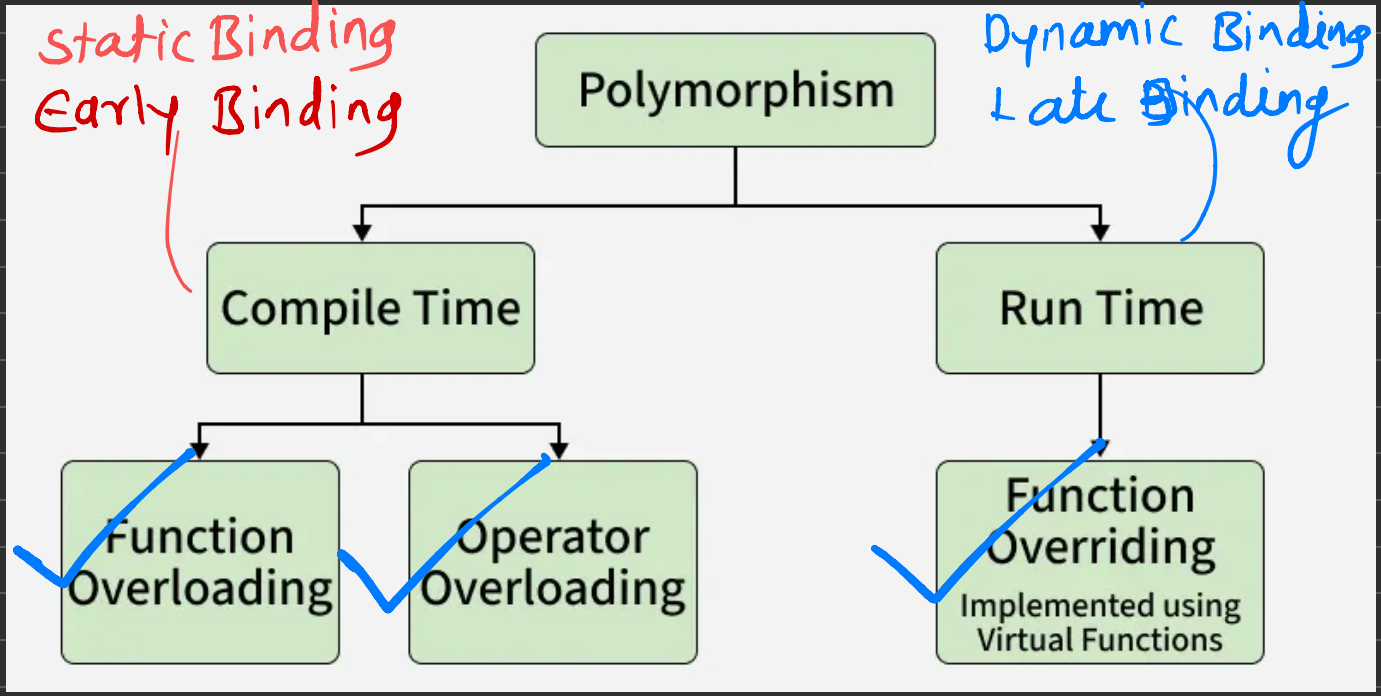
more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



Polymorphism

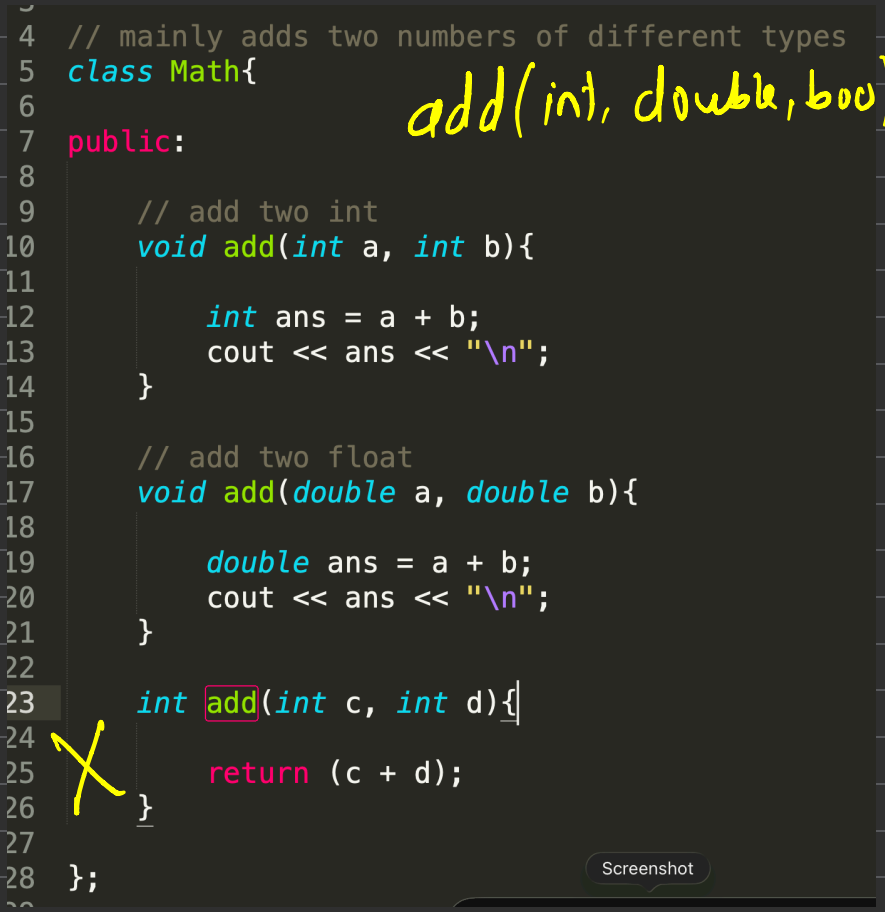
When single entity acting differently in different scenarios is called as Polymorphism.





function overloading

- Two or more functions having same name but diff parameter



add(int, double, boo) – Return type does not matter

– All methods should be in class

– parameter's data type must be diff.

Operator Overloading

int, float, char

+ - * / %, ++, --

(primitive data type

x = 10, y = 20

z = x + y

Arithmetic

minus

cannot be used on object

Math obj1

Math obj2

Math ans = obj1 + obj2 ✗

```cpp
// operator overloading
class Box{

private:
    int length;
    int width;

public:
    Box(int length, int width){

        this -> length = length;
        this -> width = width;
    }


    void show(){
        cout << "length of the box : " << length << "\n";
        cout << "width of the box : " << width << "\n";
    }
};
```

Constructor

Name of function

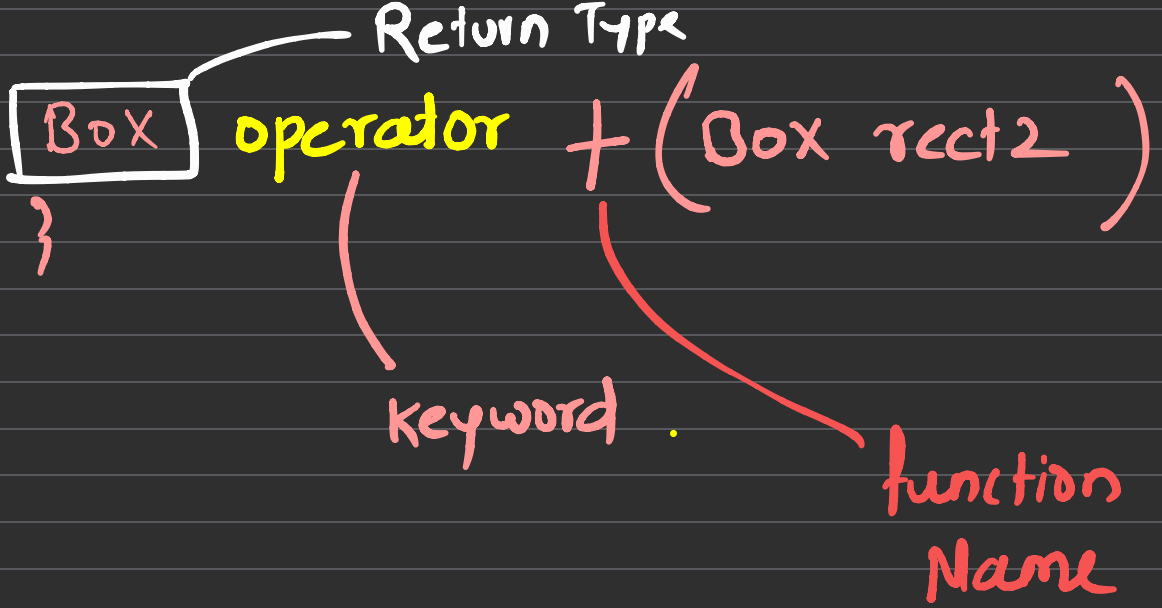
passed as parameter

Box rect1 (10, 20)

Box rect2 (30, 40)

Box ans = rect1 + rect2

Box ans = rect1 + rect2



```
Box operator + (Box rect2)
{
    int newLen =

}
```

Method Overriding

Redefining the meaning of function (method) into another class → using inheritance

Abstraction

Abstraction in [redacted] is the process of hiding internal implementation details and showing only essential functionality to the user. It focuses on what an object does rather than how it does it.

Another real life example of Abstraction is ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can't know internal details about ATM.

Real Life Example of Abstraction

OOPs

- class
- Obj
- Constructor
- this pointer
- Accessor Specifier
  - private
  - public
  - protected

} OOPs fundamental

Pillars of OOPs

- Encapsulation
- Polymorphism
- Abstraction
- Inheritance

- Single inheritance
- multilevel
- mutiple inheritance
- Hierarchical inheritance