

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

① primitive

② Non-primitive

③ 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 members
```

```
    // methods  
}
```

```
class Person  
{  
    int age  
    String nm  
  
    void show()  
    {  
        print (nm, age)  
    }  
}
```

int $x = 10$

int
(logical)
entity

physical entity

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)

}

L

Person Obj

(.) dot operator

Obj. age

Obj. nm

Obj. show()

int $x = 10$

x

`cout << x` $\rightarrow \underline{\underline{10}}$

10

obj)

`cout << obj`

age	nm
short)	

Access Specifiers *

Mainly decides the scope of accessibility of members of the class

default

public

private

Protected



(by default)

inheritance

Private .

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

Public

- Anyone can access (inside / outside)

Class

Implicit → compiler

Explicit → 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

```
class Person  
{  
    int age  
    string nm
```

public :

```
    Person()  
    {  
        age = 20  
        nm = "Sahil"  
    }
```

Name of
constructor

Person Obj

call constructor



- if no constructor is defined → then behind the scene default constructor Exist
- The moment you declare 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:  
    // parameterized 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

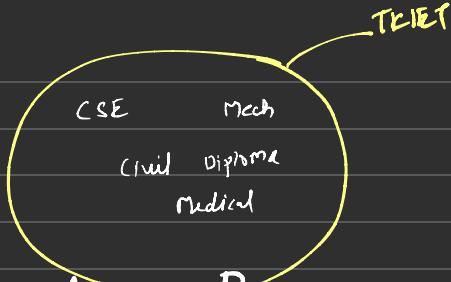
this → name = name

this → marks = marks

↳

this is class variable

Encapsulation



Binding of data member and member functions
which performs operation on data members

class Person

together in same class

{
 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 * 100
    }
}
```

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

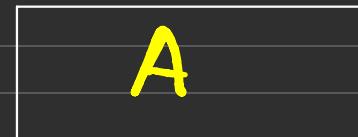
Inheritance

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

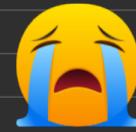
```
// Base Class  
class Person{  
  
};  
  
// Derived Class  
class Pramod: public Person{  
  
};
```

JAVA

```
// Base Class  
class Person{ 3 usages 1 inheritor  
  
}  
  
•  
  
// Derived Class  
class Pramod extends Person{ 2  
  
}
```

```
// Base Class  
class A{  
};  
  
// Base class  
class B{  
}  
  
// Derived Class  
class C: public A, public B{  
};
```

In java, we cannot extend 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 become 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 Inheritance		
	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

age

name

Base

private > protected

private > public

protected > public

Derived

private > protected > pub

Types of Inheritance

Single Inheritance

Multilevel Inheritance

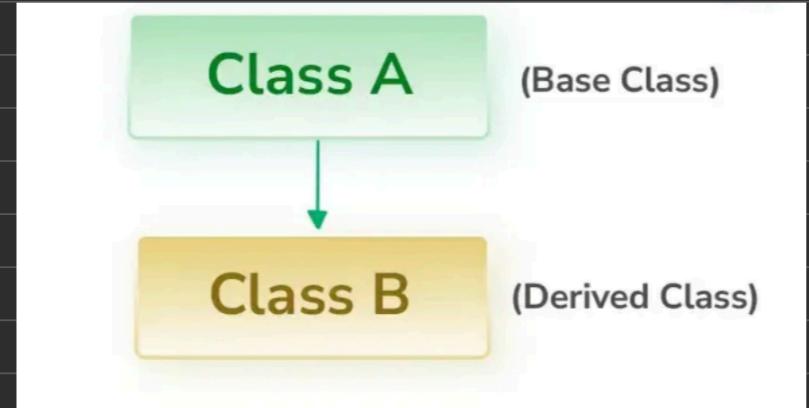
Multiple Inheritance X

Hierarchical Inheritance

Java

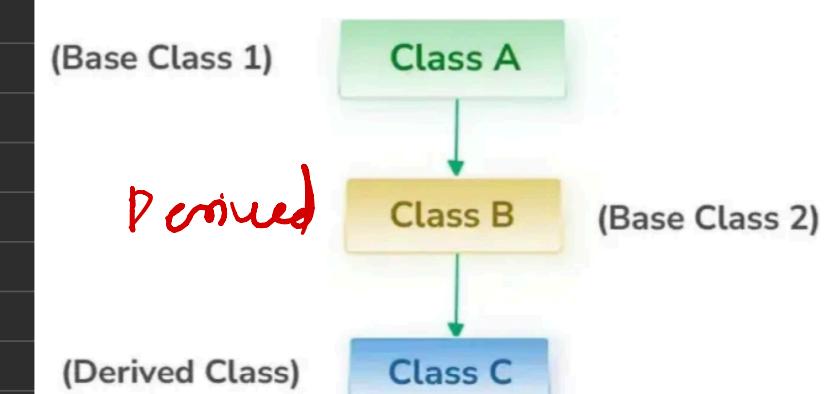
Single Inheritance

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.



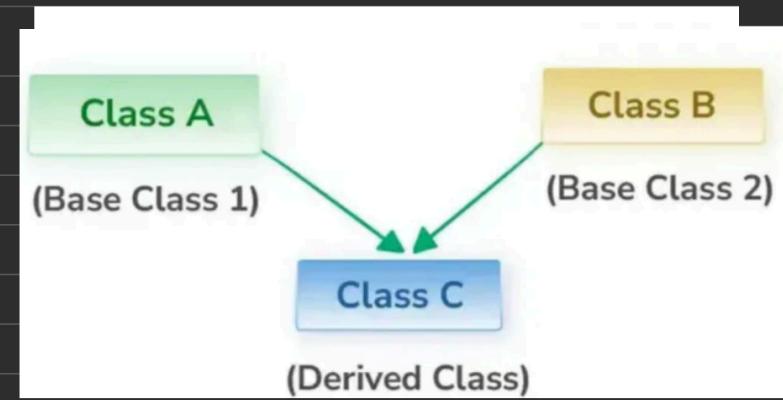
Multilevel Inheritance

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.



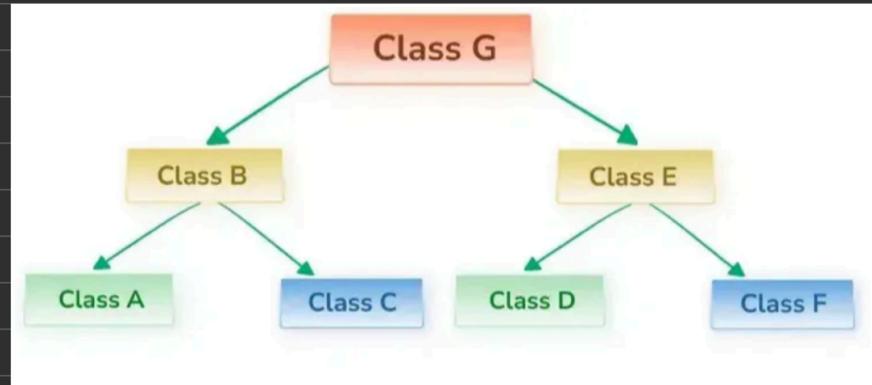
Multiple Inheritance

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.

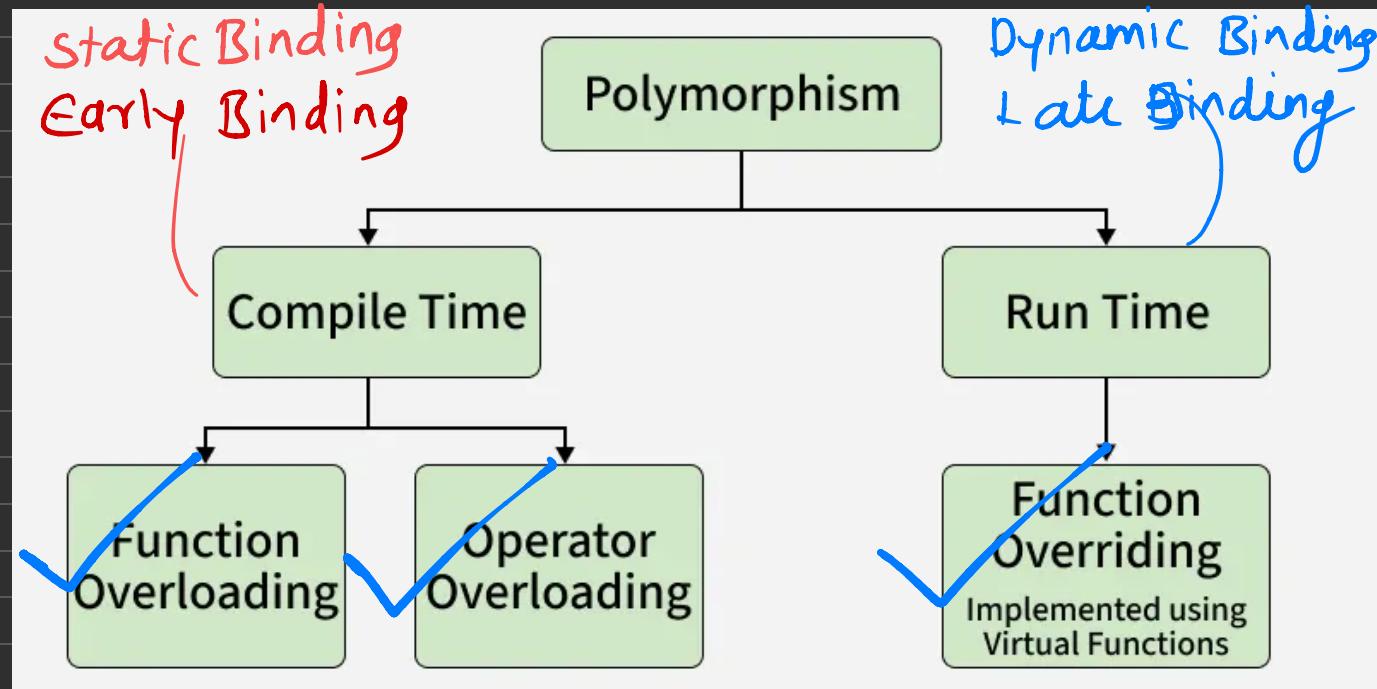


Polymorphism

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

Polymorphism

{ poly → many
morphism → form



function overloading

— Two or more functions having same name
but diff parameter

```
~ 4 // mainly adds two numbers of different types
~ 5 class Math{
~ 6
~ 7 public:
~ 8
~ 9     // add two int
~10    void add(int a, int b){
~11
~12        int ans = a + b;
~13        cout << ans << "\n";
~14    }
~15
~16    // add two float
~17    void add(double a, double b){
~18
~19        double ans = a + b;
~20        cout << ans << "\n";
~21    }
~22
~23    int add(int c, int d){
~24
~25        return (c + d);
~26    }
~27
~28 };
```

add(int, double, bool)

- Return type does not matter

- All methods should be in class

- parameter's data type must be diff.

Screenshot

Operator Overloading

int, float, char

+ - * / %. , ++, --

(primitive data type

Math obj1

Math obj2

Math ans = obj1 + obj2 X

$x = 10, y = 20$ Arithmetic
 $z = x + y$ minus

```
// 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";
    }
};
```

Box rect1 (10, 20)

Box rect2 (30, 40)

Constructor
Name of function

passed as parameter

Box ans = rect1 + rect2

Return Type
Box operator + (Box rect2)
}
Keyword .
function
Name



Box operator + (Box rect2)

?

int newLen =

↳

Method Overriding

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

Abstraction

Abstraction in Java 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

