Object-Oriented programming

→ It is a programming model which revolves around "objects".

→ objects can be considered as real-world instances of entities like class, that contain some characteristics & behaviors specified in the class template

→ class can be considered as the template, based on which object are created. (logical entity used to define a new data type)

→ object also called "instances"

→ • characteristic = 'what' about the object (data/properties)
• behaviour = 'how' about the object (functions/methods)

→ The concept of "objects" allows the oops model to easily access, use & modify the instance data & methods, interact with other objects, & define methods in runtime.

→ C++/Java/python/C#/PHP/JS

* Need for oops :) feature of data hiding

1) oops helps to understand the software easily, although they don't know the actual implementation.
2) with oops, the readability, understandability & maintaining of the code increases multifold.
3) Even very big software can be easily written & managed easily using oops.

* Note: When a class is defined, no memory is allocated, but memory is allocated an object is created.

* Some other programming paradigm/model other than OOP:

→ Methods of classification of programming lang. based on their features.

→ Mainly of two types:

1) Imperative programming model: Focus on 'How' to execute program logic & defines control flow as statements.

- → procedural programming model: Specifies the steps a program must take to reach the desired state, usually read in top to bottom.
- → object oriented programming: Organise programs as objects contain data & behaviours
- → parallel programming: It breaks a task into subtasks & focuses on executing them simultaneously at same time.

2) Declarative programming model: Focus on 'what' to execute & defines program logic, but not a detail control flow

- → Logic programming model: (formal logic) Refers to set of sentences expressing facts & rules abt. how to solve problem.
- → functional programming model: program constructed by applying & composing functions
- → Database programming model: It used to manage data & information

* Advantages of using oops :

1) Helpful in solving very complex level of problems.
2) oops, promote reuse code, thereby reducing redundancy
3) Hide unnecessary details with help of data Abstraction
4) oops uses bottom up approch (unlike structural programming)

* Access specifiers : sets some restrictions on class members from accessing the outside function directly.

→ vital Role in securing data from unauthorized access.

	Inside class	child (Sub-class)	Outside class
Public	✓	✓	✓
protected	✓	✓	x
private	✓	x	x

member function inside the class

* Disadvantages of oops:

1) Require pre-work & proper planning
2) In certain scenarios, program can consume a large amount of memory.
3) Not suitable for small problems
4) proper documentation required for later use

* Difference b/w class and structure : having two differences

→ The most important of them is hiding implementation details.

Class (save in heap)	structure (stack)
1) private by default	public by default
2) class keyword	struct keyword
3) reference type	value type
4) use data abstraction & further inheritance	uses grouping of data
5) Can have Null values	Can't have Null values
6) May have all types of Constructors	May have only parametrized constructor

* Difference b/w Constructor and Methods :

Constructor	Method
1) Block of code which initialize a newly created object	2) group of statements that can be call at any pt. in the program using its name to perform specific tasks.
2) Same name as class name.	2) Should have different name than the class name.
3) It has no return type.	3) void / should return a value.
4) Called implicitly at time of creation	4) explicitly by programmer
5) Default Constructor is there	5) no default method is provided.

* Constructor :

→ special member function automatically called when an object created

→ For every object in its lifetime constructor is called only once at time of creation

→ has same name as class

→ doesn't have any return type

→ It is public

→ Types of Constructor : 1) Default

•) When we write our constructor explicitly, the inbuilt constructor will not available for us.

2) parametrized constructor :

```
Public :
    class_name(int num, string str) {
        type1 = num;
        type2 = str;
    }
```

3) Copy Constructor : Constructor that takes an object as an argument & copies values of one object's data member into another object

→ we have to pass object's name whose values we want to copy, & when we are passing an object to Constructor, we must use (&) operator.

↳ Constructor overloading : differ the no. of arguments.

* Destructor: Delete the object.

→ Purpose is to free the resources that the object may have acquired during its lifetime.

→ If object is created by using new keyword / dynamically we have to use "delete" keyword

→ we can have one destructor in a class

→ Can be declare in private

* this pointer: hold current object's address

→ 3 Main usages:

1) Can be used to refer to a current class instance variable.
2) Can be used to pass current object as a parameter to another method.
3) Can be used to declare indirect.

* Note: this pointer accessible only within non-static function

* Shallow Copy: object is created by simply copying the data of all variables of the original object.

→ Here the pointer will be copied but not the memory

→ It means that the original object & created copy will now point to same memory location, which is not prefered.

Note: C++ compiler implicitly creates a copy constructor & assignment operator to perform shallow copy at compile time.

* Deep copy:

Note: An object is created by copying all fields, & it also allocates similar memory resources with the same value to object.

→ we need to explicitly define the copy constructor & assign dynamic memory as well if required.

→ It is also necessary to allocate memory to other constructors variable dynamically.

→ Shallow copy

1) Stores references of objects to original memory address.

2) It reflects changes made to new/copied object in original object

3) faster

Deep copy

1) Stores copies of the object's value.

2) It doesn't reflect

3) slower

→ student (student const &s) { → avoid ∞ loop (reference) / can't change value

this.age = s.age

this.name = new char [strlen(s.name) +1]

strcpy (this→name, s.name);

Copy Constructor

* Initialization list: Memory allocate to const variable at the time of memory allocation

```
class Student {
Public:
const int rollno;
student (int r): rollno(r), {
}
```

rollno → object; r → var; other variables (not use this)

* Constant functions: From Constant function you can only call constant functions

↳ which doesn't change any property of current object

* operator overloading: operator extends our pre existing functionality, such that it will work for our user define classes.

eg: Fraction operator +(Fraction const &f2){

. . . .

}

f4 = f1 + f2

(argument)

this (automatically)

: Equal to operator for fraction class

bool operator ==(Fraction f2){

return (num == f2.num && deno == f2.deno);

}

→ '+' operator overloading:

pass by reference illegal changes not allow

Fraction operator +(Fraction const &F2) const{

not change in this

int lcm = deno * f2.deno;

int x = lcm/deno;

int y = lcm/f2.deno;

int a = x * num + (y * f2.num);

Fraction FNew(a, lcm);

fNew.simplify();

return fNew;

}

* 'pre increment operator':

```
Fraction& operator++() {
    num = num + deno;
    Simplify();
    return *this;
}
```

must by reference

* 'post increment operator':

```
Fraction operator++(int) {
    Fraction fNew(num, deno);
    num = num + deno;
    fNew.simplify();
    return fNew;
}
```

- Dynamic array class: (arr, nxtidx, capacity)

Encapsulation: Refers to bundling data and methods that operate on that data into a single unit.

eg: class

→ Restricts direct access to some components of an object, s.t. users can't access state values for all variables of a particular object.

Define in 2 way

Data hiding

data binding

Abstraction: It means providing only some of the info. to user by hiding its internal implementation details.

→ we just need to know about the method of the objects that we need to call & input parameters needed to trigger a specific operation, excluding the details of implementation & type of action performed to get result.

→ Access specifiers are the main pillar of implementing abstraction in C++.

→ eg: Tv Remote function: we only press buttons like (volume up/down) without bothering of implementation part

→ Helps in Reducing programming complexities & efforts.

→ Make application secure

→ increase reusability of code.

→ Avoid duplication

* Why use abstraction:

1) updation/changes doesn't affect user's output

eg: sort() → Bubble sort } same output
→ Merge sort

2) Avoid the errors

Inheritance: Allows us to create a new class (derived class) (subclass) from an existing class (base class) (superclass)

→ time taking & Derived & base class tightly bound

→ The derived class inherits the features from base class and can have additional feature of its own

→ Modes of inheritance:

Base class member access specifier	Public	protected	private
public	public	protected	private
protected	protected	protected	private
private	~~protected~~ not access	not access	not access

* Syntax : class parent_class { };

class child_class : access_modifier parent_class { };
↳ by default private

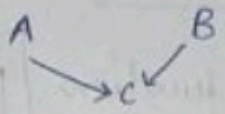
Types of Inheritance :

1) Single Inheritance : class A (one parent) ↓ class B (one child) { 1 Level

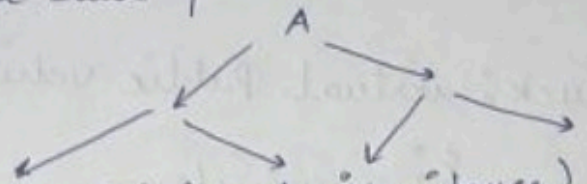
2) Multilevel Inheritance : more than one level

class A
↓
class B
↓
class C

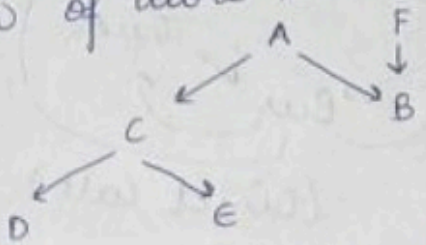
3) Multiple Inheritance :
class can inherit more than one class.



4) Hierarchical Inheritance :
one class is base class for more than one derived class



5) Hybrid Inheritance : (virtual inheritance)
Combination of more than one type of inheritance



* Order of Constructor : A a; A()
↓ B b; A() B()
↓ C c; A() B() C()

• Destructor : Order reverse

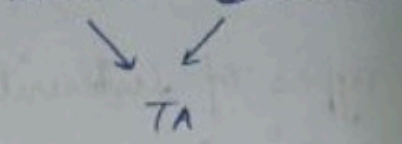
→ By default calling base class constructor, if want to pass paramdrized constructor then → initialization list

eg : car() : vehicle(10) {
//
}

* In multiple Inheritance, parent have same function then child class calling that function give error to resolve

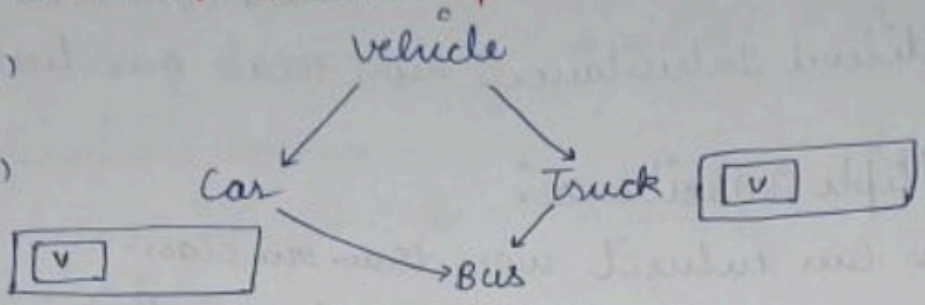
Syntax:

```
Ta a;
a.Student :: print();
```



FAMOUS PROBLEM: Dimond problem

Calling: Vehicle()
Car()
Vehicle()
Truck()
Bus()

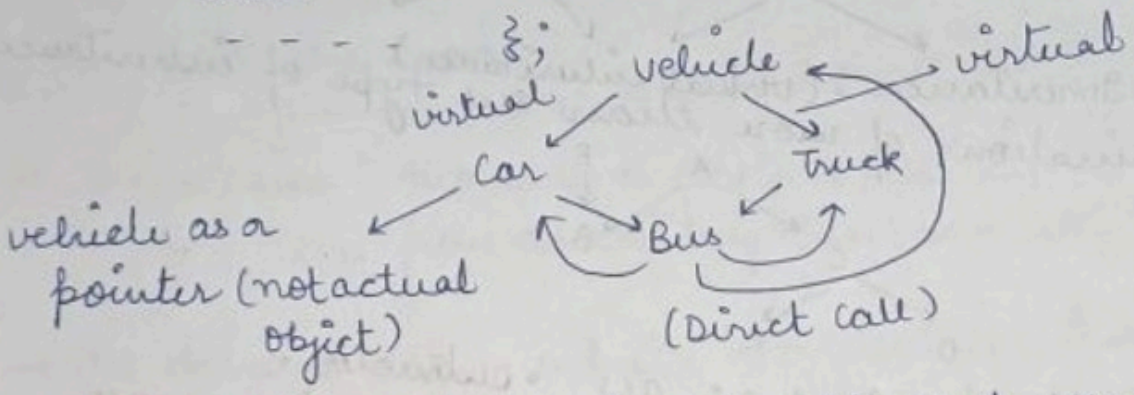


→ Copy of virtual functions

To avoid call virtual

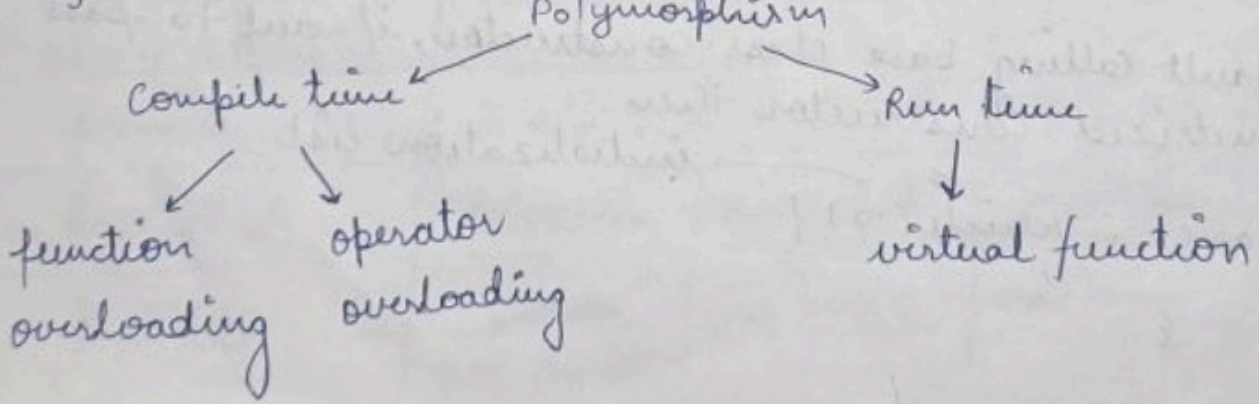
Syntax:

```
Class Truck: virtual Public vehicle{
- - - -
};
```



Polymorphism: It is a concept that allows you to perform a single action in different ways. (Many forms)

eg: A person can be son, father, husband etc.



* compile time polymorphism : (static polymorphism)

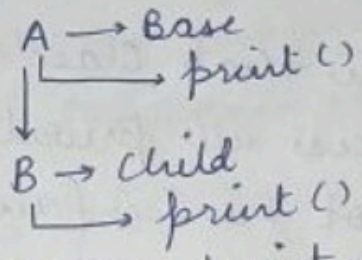
1) Function overloading : Multiple function with same name but different parameters.
   → Increases program's readability

2) operator overloading : operator behaves differently in different ways.
   eg: '+' → 1+2=3 | 1.5+2.7=4.2 | 'a'+'b'=97+98

* point to Remember
  • precidence & associativity of operators remain intact
  • = and & already overloaded in C++
  • not for inbuilt operators

→ Can't be overloading = ::|.|.*|?:

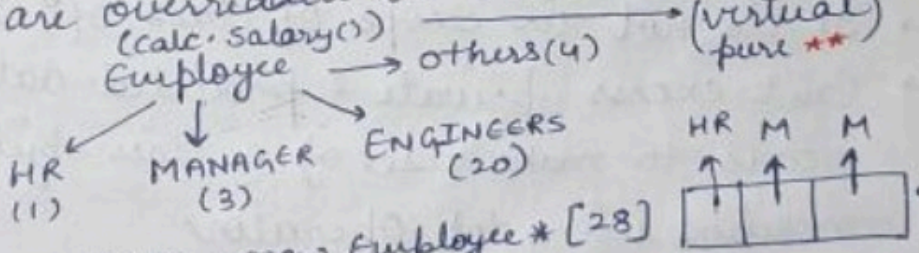
3) Method overriding :



Note: we can point parent class pointer to child class but not vica-versa
→ excess only those property which is in parent class

* Run time polymorphism : (dynamic polymorphism)
→ virtual functions : functions which are present in a base class & they are overridden in derived class.

eg: Calculating salary.



CalcSalary(); Employee **e = new Employee*[28]
e[i] → CalcSalary();
↓
must be run time

Pure virtual function:

virtual void A() = 0;

→ If in a class atleast one pure virtual function present then that class is called Abstract class
  → not able to create object

vehicle (Abstract)
↓
Car ——→ implement all pure virtual fu.
  └——→ become an abstract

Friend functions: If a function is defined as a friend function in c++, then the protected & private data of a class can be accessed using the function

→ define outside that class's scope
→ Right to access all private & protected members
→ friends are not member functions

Syntax: class class_name { → global
  friend data_type func_name(arg);
};

→ Can be declare in private or public section
→ normal function without using object
→ It is not in scope of class, of which it is a friend
→ Can't excess private & protected data member directly. It needs to make use of a class object & then access the members using dot operator

→ It can be global function or a member of another class.

* friend data_type class_name::func_name();

→ not excess to this

* Is it necessary to create objects from class?

→ NO. An object is necessary to be created if the base class has non-static method.

* What is an Interface:

→ special type of class, which contains methods, but not their difination. Only the declaration of methods is allowed inside an interface.

To use an interface, you can't create objects. Instead, you need to implement that interface & define the methods for their implementation.

* Difference b/w overloading & overriding?

→ overloading: It is a compile-time polymorphism in which an entity has multiple implementations with same name. eg → function ~~overriding~~ overloading

→ overriding: It is a Run-time polymorphism feature in which an entity has same name, but its implementation changes during execution.

* Data Abstraction is accomplished with help of abstract methods or abstract classes

→ Abstract Class: An abstract class is a special class containing abstract methods. The significance of abstract class is that the abstract method inside it are not implemented & only declared. When a subclass inherits abstract class & needs to use it abstract method, they need to define and implement them.

* Java applications are based on oops, they can't be implemented without it.

→ C++ can be implemented without oops, as it supports C-like structural programming model.

* Difference b/w Interface & Abstract class?

→ Interface: Subclass must define all its methods & provide its implementation

→ Abstract class: Subclass need not to provide the definition of its abstract class, until & unless the subclass is using it.

* Smalltalk → 1st language to developed as purely oops (Alan Kay)

* Java not support all inheritance types.

What is an exception: A problem or error arises during the execution of a program. There could be errors that cause the programs to fail or certain conditions that lead to error.

→ If these run time errors are not handled by program, OS handle them & program terminates abruptly.

Eg: Errors: Divide by 0, out of bound idx

Exception Handling: It is build upon 3 keywords

→ try, catch & throw

1) Throw: Throw exception when a problem shows up.

2) Catch: Catches an exception with an exception handler at the place in a program where you want to handle the problem.

3) try: A try block identifies a block of code for which particular exceptions will be activated. followed by one or more catch blocks.