OOPS → i) Abstraction
ii) Encapsulation
iii) Inheritance -
iv) Polymorphism -

*) Encapsulation : binding together the data & the function that manipulate or access them
(done via classes)
↓
for example : student

Name rollno. age address
setRollNo() changeAdd()

*) Abstraction : Hiding ~~unnecessary~~ unnecessary details, done using access specifiers, and protecting data.
from illegal tampering.
↓
i) TV & remote example
ii) inbuilt sort function ← internally, which algorithm is used, is not of concern
iii) Standard template library

Access specifiers : i) private : only member functions can access these attributes.
ii) public : any one can access the data member
iii) protected : only accessible by child classes / derived classes.

Scanned by CamScanner

•) Internal changes to the abstracted entity does not result the functioning of the use case, since dependency does not get affected.

*) Inheritance : inheriting properties & methods from parents, with our own specific properties & methods.

↓

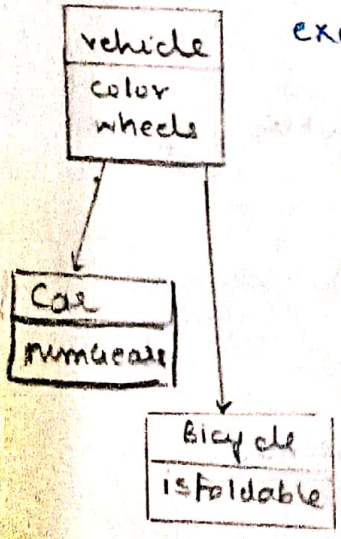
need : birds with different specific fly method.

↓

reducing code redundancy, and increasing efficiency & readability

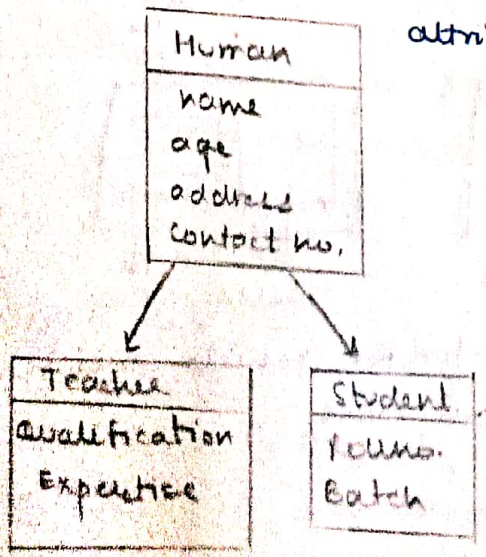
↓

Code reusability



example: i) vehicles is the parent class to car, bicycle or truck with common properties like color, no. of wheels, etc. and different specific properties. so the common properties & methods can be inherited from a more generalised class like vehicle, and specialisation can be added on the current level.

ii) Teacher & student classes would have some common attributes, that can be attributed to being a person/human & hence this order of inheritance, where human adds a sense of generalisation and teacher & student add their own specific level of speciality.



Scanned by CamScanner

Inheritance syntax : class derived-class-name
: access-specifier base-class-name { };

↓

class car : public vehicle { };

** private → x
protected → protected
public → public

↓ inheritance done using protected

private → x
protected → protected
public → protected

↓ inheritance done using public

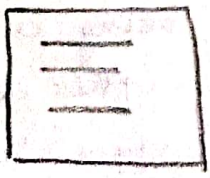
private → x
protected → private
public → private

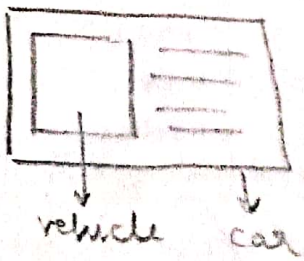
↓

By default, access modifier is assumed to be private

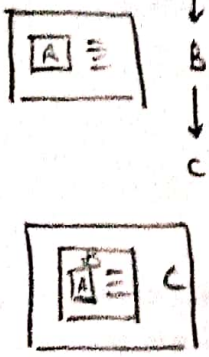
*) order of constructor / destructor calls.

Since the car class inherits members from its parent class & has members of its own as well. Therefore the constructor for vehicle class is called first & then the one for car class.





Scanned by CamScanner



Aa; → AC()

Bb; → AC()
↓
BC()

Cc; → AC()
↓
BC()
↓
CC()

Constructor's for ancestor classes are called implicitly & are not required to be done explicitly.

↓

Destructors are called in opp. order

initialisation list : doing something before entering the function.

↓

By default, the default constructor is called of the parent class.

↓

If there is another parameterised available, then it can be called using initialisation list

↓

Car() : vehicle(5) {}

↓

If the default constructor of the base class is not visible, then the parameterised will be tried at, but if no parameterised or explicit calls are marked in the initialisation list, till then error is produced.

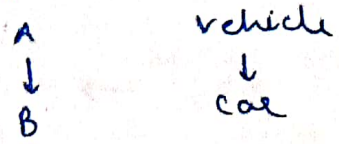
↓

Car(int x) : vehicle(x) {} → right way to go about it.

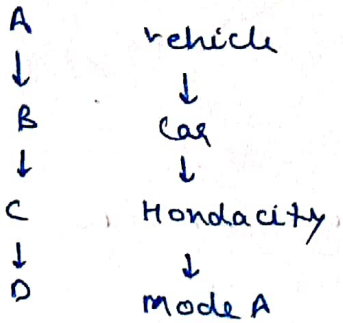
* for any class, it can only call the constructor of its immediate base class & not above it.

Scanned by CamScanner

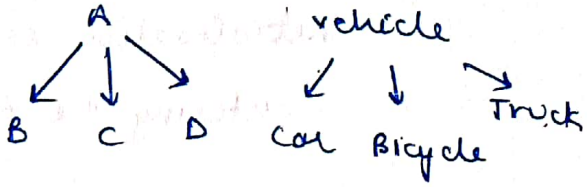
*) Type of inheritance : (i) single inheritance



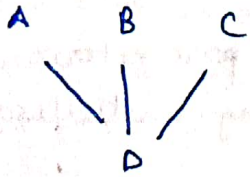
(ii) multilevel inheritance :

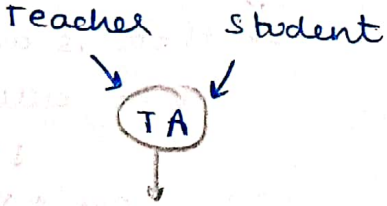


(iii) Hierarchical inheritance :

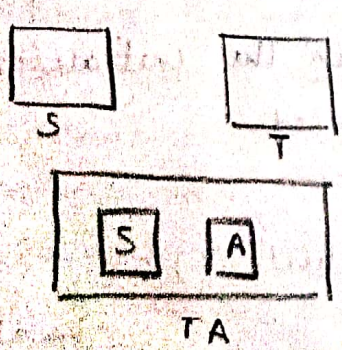


(iv) Multiple inheritance : Multiple base classes for one derived class.





class TA : public Teacher, public Student { };



constructor calls will take place in order of specification in definition for same level.

Scanned by CamScanner

If there is a common fn in both base classes, then ambiguity is resolved using scope resolution

↓

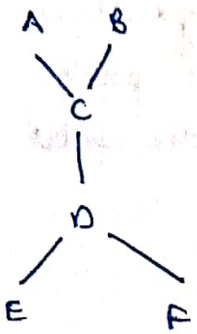
a.Student :: print( );

↓ similarly for data members/attributes

a.Teacher :: expertise;

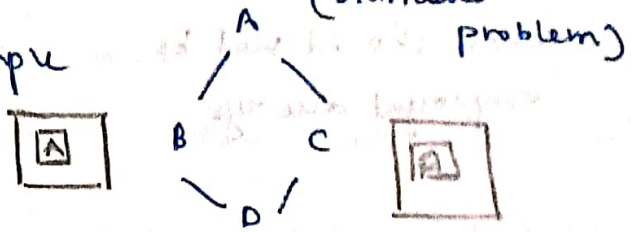
Also called virtual inheritance

v) Hybrid inheritance → mixed inheritance of different kind to form a composite structure

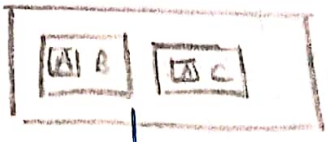


↓

for example



(Diamond problem)



↓

has will have ambiguity due to multiple copies of members & methods of A.

↓

can be resolved using 2 methods:

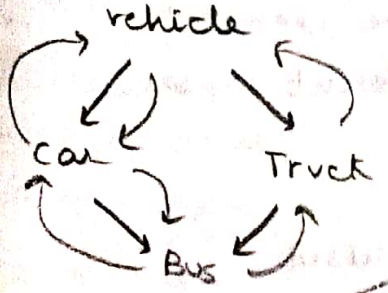
i) method overriding in current class.

ii) scope resolution from Car or Truck's vehicle inheritance

↓

which ultimately calls vehicle's print fn

↓ the copying or creation of same class can be stopped using virtual keyword with inheritance mechanism.



vehicle()
Car()
vehicle()
Truck()
Bus()

b. Car :: print()

b. Truck :: parent()

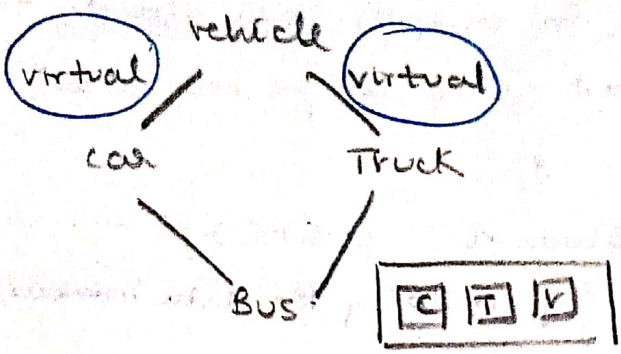
class Truck : virtual public vehicle

{

};

Now, Bus will directly call vehicle constructor, rather than Car & Truck calling it

Scanned by CamScanner



vehicle()
car()
truck()
Bus()

virtual base class → necessary care is taken so that duplication is avoided regardless of the number of paths that exist to the child class.

↓

a heavy check mechanism is involved for this, hence should not be employed always.

*) Polymorphism → many forms

↓

set of code to be executed depends on different contexts involved.

↓

i) Compile time polymorphism
ii) Runtime polymorphism

i) Compile time : is used in a) operator overloading (1) + for int, double
b) function overloading (2 or more constructors for a class)
c) method overriding.

b) int test (int a, int b){}
int test (int a){}
int test (){}

→ apt function called depending on the call being made

↓

example of function overloading

* does not work with different return type ←

Scanned by CamScanner

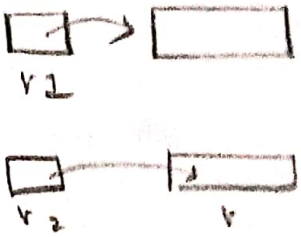
c) The child class redefines a function defined in the base class is called method overriding

↓

compile time because, the calling is decided at compile time itself for that specific object.

↓

```
vehicle * v1 = new vehicle();
vehicle * v2;
v2 = &v;
v2 = &c;
```



→ ✓ (Base class pointer pointing to child class object)

↓

(vice-versa not allowed)

↓

Base class pointer can be used to access only properties/methods defined in base class.

(type of ptr)

Compile time polymorphism

function using pointer of class, and accordingly method is used

↓

This can be replaced by runtime polymorphism, by using virtual keyword, and this can be used to decide the call depending on type of object being referred to by the pointer.

(content of ptr)

↓

method is defined at run-time

↓

virtual functions (method, we expect to be overridden)

↓

use case: organisation has different kind of employees, and their salary needs to be computed, and salary calculation is dependent on their kind.

Scanned by CamScanner

(1) (3) (20) (4)

for example : HR, Managers, Engineers, others

↓

employee class is created, and accordingly derived class are created for each type.

↓

28 employees → have a same name function called calculate salary().

↓

rather than having to loop on different types separately, we operate using the parent / base class pointers to make this much more easier.

↓

an array of employees can be created, and according to type, calcsalary() can be called.

```
Employee ** e =
    new Employee * [28];
e[i]. calcsalary();
```

↓

virtual function present, and hence runtime polymorphism gets involved.

*) virtual fns & abstract classes → analogous to an interface in JAVA

↓

pure virtual fn

↓

no definition

↓

virtual void print() = 0;

↓

class having atleast one pure virtual fn.

↓

can only be inherited & not instantiated

↓

i.e. needs to overridden in derived class, i.e. derived class should either implement all pure virtual fns, or become an abstract class

← concrete class

Scanned by CamScanner

*) enforces specific definition for derived classes.

*) friend functions & classes

↓

can be employed to access private and protected properties / methods of a given class.

↓

needs to be marked as friend in the class whose private members need to be accessed.

↓

friend void Bus :: print ( ) ; → not member functions,

↓

this class needs to be defined before this class

↓

access specifiers don't affect behaviour at all.

↓

does not have access to "this";

↓

execution : main

compilation : start of document

↓

void print ( ) ; → function declaration

void print ( ) { } → function definition

↓

void Bus :: print ( ) { }

↓

scope resolution

↓

friend class Bus ; → works for entire class

↓

each member of the class Bus can access truck private members.

↓

both need to mark each other as friend. ← one way

Scanned by CamScanner