OBJECT ORIENTED PROGRAMMING -

#OOP - is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts defined below:

Class - is a user-defined data type which defines its properties and its functions. Class is the only logical representation of the data. For eg- Human being is a class. The body parts of a human beings are its properties, and the actions performed by the body parts are known as functions. The class does not occupy any memory space till the time an object is instantiated.

```
class student {
    public:
        int id;          → data member (State)
        int mobile;
        int add() { → member function. (Behaviour)
            return 5;
        }
};
```

A class is like a blueprint for an object.

→ physical entity having state and behaviour

Object - is a run-time entity. It is an instance of the class. An object can represent a person, place or any other item. An object can operate on both data members and member functions.

Student s = new student(); → object of student

Note - When an object is created using a new keyword then space is allocated for the variable in a heap, and the starting address is stored in the stack memory. When an object is created without a new keyword, then space is not allocated in the heap memory, and the object contains the null value in the stack.

Characteristics of an Object Oriented programming:-

Inheritance - is a process in which one object acquire all the properties and behaviours of its parent object automatically. In such a way, you can reuse, extend or modify the attributes and behaviors which are defined in other classes.

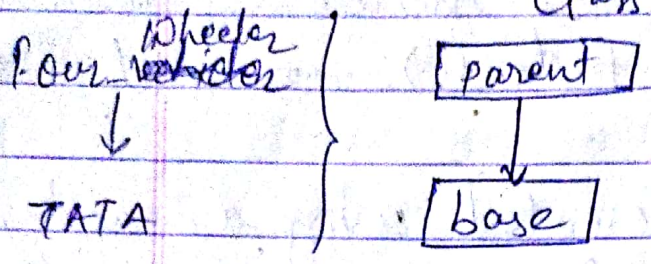
In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

class derived-class : visibility base-class;

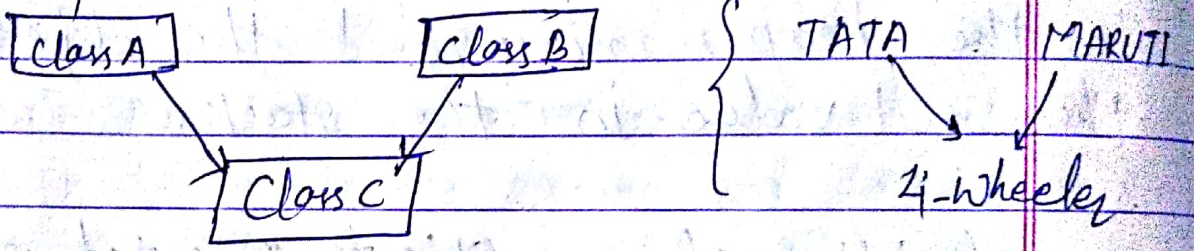
Scanned by CamScanner

Types of Inheritance:

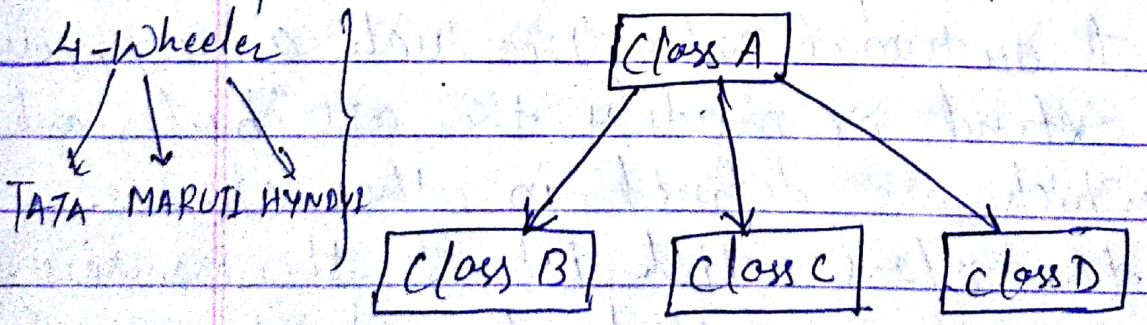
1) Single inheritance: When one class inherits another class,



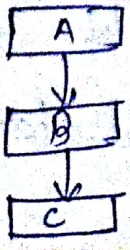
2) Multiple inheritance: is the process of deriving a new class that inherits the attributes from two or more classes.



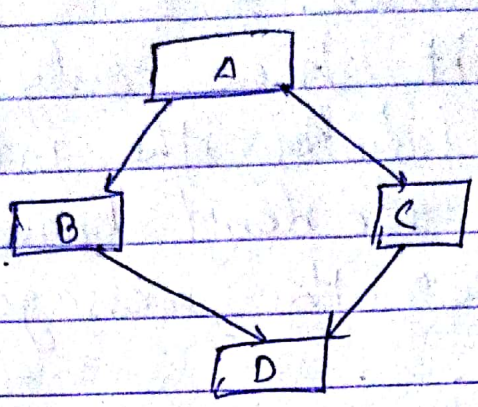
3) Hierarchical inheritance: is defined as the process of deriving more than one class from a base class.



4) Multilevel inheritance- is a process of deriving a class from another derived class.



5) Hybrid inheritance - is a combination of simple, multiple inheritance and hierarchical inheritance.



Encapsulation - Encapsulation is the process of combining data and functions into a single unit called class.

→ In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class.

→ In simple words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attribute

→ Thus, encapsulation makes the concept of data hiding possible.

Data hiding - A language feature to restrict access to members of an objects, reducing the negative effect due to dependencies

e.g.- "protected", "private" feature in C++.

Abstraction - process of hiding the implementation details and showing only functionality to the user.

→ Another way, it shows only essential things to the user and hides the internal details.

E.g:- Sending SMS, we don't know the internal processing about the message delivery.

Data binding - is a general technique that binds data sources from the provider and consumer together and synchronizes them.

Polymorphism - polymorphism is the ability to present the same interface for differing underlaying forms.

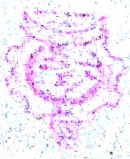
→ polymorphism means having many forms.

→ polymorphism means the ability of a message to be displayed in more than one form.

E.g:- A person at the same time can have different characteristics. Father, son, husband etc.

Types of polymorphism -

1) Compile Time polymorphism (static)
2) Runtime polymorphism (Dynamic)



(static)

- **Compile Time polymorphism:** The polymorphism which is implemented at the compile time is known as compile-time polymorphism

example-

↓ (same name but different parameter)

Method overloading: Method Overloading is a technique which allows you to have more than one function with the same function name but with different functionality. Method overloading can be possible on the following basis:

1. return type of the overloaded function.
2. The type of the parameters passed to the function.
3. no. of parameters passed to the function.

Operator overloading: is a way of providing new implementation of existing operators to work with user-defined data type.

(dynamic)

- **Runtime polymorphism:** To call the function is determined at runtime is called runtime polymorphism.

e.g. - **Function overriding** - means when the child class contains the method which is already present in the parent class. Hence, the child class overrides the methods of the parent class. In case of funct^n overriding, parent and child class both have same function

Page No. .....
Date .....

with a different definition.

Constructor – Constructor is a special method which is invoked (call) automatically at the time of object creation. It is used to initialize the data members of new objects generally.

→ The Constructor in C++ has the same name as class or structure.

There can be two types of Constructor in C++

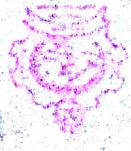
1) Default Constructor - A Constructor which has no argument is known as default.

→ It is called at the time of creating an object.

2) Parameterized Constructor - Constructor which has parameter is called a parameterized constructor.

→ It is used to provide different values at distinct objects.

3) Copy Constructor - A copy constructor is a overloaded Constructor used to declare and initialize an object from another object. It is of two types - default copy constructor and user defined copy Constructor.

E.g:-
```
class go{
    public:
        int x;
        go(int a){ → parameterized constructor
            x=a;
        }
        go(go &i){ → Copy constructor
            x=i.x;
        }
        go(){ → Default constructor
            y=5;
        }
};
int main()
{
    go a1(20); → calling parameterized const.
    go a2(a1); → calling copy constructor.
    return 0;
}
```

Destructor: A destructor works opposite to constructor it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically. A destructor is defined like a constructor. It must have the same name as class, prefixed with a tilde sign (~).

E.g.:-
```
Class A{
    public:
        A(){
            cout<<"Constructor called"<<endl;
        }
        ~A(){
            cout<<"destructor called"<<endl;
        }
};
int main()
{   A a;
    A b;
}
```
⟶ Constructor called
Constructor called
Destructor called
Destructor called

"this" pointer- this is a keyword that refers to the current instance of the class. There can be 3 main uses of 'this' keyword:

1. It can be used to pass the current object as a parameter to another method.
2. It can be used to refer to the current class instance variable.
3. It can be used to declare indexers

E.g:-
```
Struct node {
    int data;
    node *next;
    node (int x){
        this->data=x;
        this->next = NULL;
    }
};
```

#Friend Function - Friend function acts as a friend of the class. It can access the private and protected members of the class. The friend function is not a member of the class, but it must be listed in the class defination. The non-member function cannot access the private data of the class.

→ Sometimes, it is necessary for the non-member function to access the data. The friend function is a non-member function and has the ability to access the private data of the class.

Note - 1) A friend function cannot access the private members directly, it has to use an object name and dot operator with each member name.

2) Friend function uses objects as arguments.

E.g:-

```
class A{
    int a=2;
    int b=4;
  public:
    friend int mul(A K){ ⟶ friend function
      return (K.a * K.b);
    }
};
int main(){
    A obj;
    int res = mul(obj);
    cout<< res << endl;
    return 0;
}
```

⇒ output = 8.

Aggregation: It is a process in which one class defines another class any entity reference. It is another way to reuse the class. It is a form of association that represents the HAS-A relationship.

Virtual function — A virtual function is used to replace the implementation provided by the base class. The replacement is always called whenever the object in question is actually of the derived class, even if the object is accessed by a base pointer rather than a derived pointer.

1) A virtual function is a member function which is present in the base class and redefined by the derived class.

2) When we use the same function name in both base and derived class, the function in base class is declared with a keyword virtual.

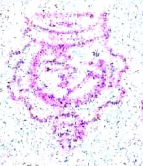
3) When the function is made virtual, then C++ determines at run-time which function is to be called based on the type of the object pointed by the base class pointer. Thus, by making the base class pointer to point to different objects, we can execute different versions of the virtual functions.

Key points: 1) Virtual functions cannot be static.

2) A class may have a virtual destructor but it cannot have a virtual constructor.

E.g:-

```
class base{
    public:
        virtual void print(){
            cout<<"print base class"<<endl;
        }
        void show(){
            cout<<"show base class"<<endl;
        }};
```



```
class derived : public base{
  public:
        void print(){
            cout<<" print derived calls";
        }
    void show(){
        cout<<" show derived class";
    }};
int main()
  { base *ptr;
    derived d;
    ptr = &d;
    ptr -> print();
    ptr -> show();
  }
```

⇒ output - print derived class (due to Virtual function)
Show base class

Pure Virtual Function -

1. A pure virtual function is not used for performing any task. It only serves as a placeholder.
2. A pure virtual function is a function declared in the base class that has no definition relative to the base class.
3. A class containing the pure virtual function cannot be used to declare the object of its own, such classes are known as abstract base classes.

Page No ..... Date .....

4. The main objective of the base class is to provide the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.
e.g- virtual void display()=0;

Abstract classes: In C++ class is made abstract by declaring at least one of its functions as a pure virtual function. A pure virtual function is specified by placing "=0" in its declaration. Its implementation must be provided by derived classes.

e.g:-
```
class Shape{
    public:
        virtual void draw()=0;
};
class Rectangle : Shape{
    public:
        void draw(){
        cout<<"Rectangle";
}};
class Square : Shape{
    public:
        void draw(){
        cout<<"Square"<<endl;
}};
int main(){
    Rectangle rec;
    Square Sq;
```

```
    rec.draw();
    sq.draw();
    return 0;
}
```
⇒ output - Rectangle
Square.

Namespace in C++; 1) The namespace is a logical division of the code which is designed to stop the naming conflict.

2) The namespace defines the scope where the identifiers such as variables, class, functions are declared.

3) The main purpose of using namespace in C++ is to remove the ambiguity. Ambiguity occurs when a different task occurs with the same name.

4) For example: if there are two functions with the same name such as add(). In order to prevent this ambiguity, the namespace is used. Functions are declared in different namespace.

5) C++ consists of a standard namespace, i.e., std which contains inbuilt classes and functions. So, by using the statement "using namespace std;" includes the namespace "std" in our program.

e.g:-
```
namespace Add{
    int a=5, b=5;
    int add(){
        return(a+b);
    }}
int main(){
    int res = Add::add(); → accessing the funct inside namespace
    cout<<res;
} ⇒ output: 10
```

Access Specifiers - The access specifiers are used to define how functions are variable can be accessed outside the class. There are three types of access specifiers:

1) Private - Functions and Variables declared as private can be accessed only within the same class, and they cannot be accessed outside the class they are declared.

2) public: Functions and Variables declared under public can be accessed from anywhere.

3) protected: - functions and variables declared as protected cannot be accessed outside the class except a child class. This specifier is generally used in inheritance.

Date .....
Page No .....