What is Object Oriented Programming?

Object Oriented programming (OOP) is a programming paradigm that relies on the concept of **classes** and **objects**.

It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects.

There are many object-oriented programming languages including JavaScript, C++, Java, and Python.

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

Object is a basic unit of Object Oriented Programming and represents real life entities.
Object is an instance of a class.

For example, say we created a class, Car, to contain all the properties a car must have, color, brand, and model. We then

create an instance of a `Car` type object, `myCar` to represent my specific car.

We could then set the value of the properties defined in the class to describe my car, without affecting other objects or the class template.

We can then reuse this class to represent any number of cars.

4 pillars of OOP

Pillar 1: Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.

For example, when you login to your Amazon account online, you enter your user_id and password and press login, what happens when you press login, how the input data sent to amazon server, how it gets verified is all abstracted away from you.

Pillar 2: Encapsulation

The process of binding data and corresponding methods (behavior) together into a single unit is called **encapsulation .** It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which they are declared.

prevention of data direct access by the program is called data hiding or information hiding

Suppose you have an account in the bank. If your balance variable is declared as a public variable in the bank software, your account balance will be known as public, In this case, anyone can know your account balance. So, they declare the balance variable as private for making your account safe, so that anyone cannot see your account balance.

The person who has to see his account balance, will have to access only private members through methods defined inside that class and this method will ask your account holder name or user Id, and password for authentication.

Pillar 3: Inheritance

Inheritance is a way to reuse once written code again and again. The class which is inherited is called the Base class & the class

which inherits is called the Derived class. They are also called parent and child classes.

So when a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

For any bird, there are a set of predefined properties which are common for all the birds and there are a set of properties which are specific for a particular bird. Therefore, intuitively, we can say that all birds inherit common features like wings, legs, eyes, etc. Therefore, in the object-oriented way of representing the birds, we first declare a bird class with a set of properties which are common to all the birds. By doing this, we can avoid declaring these common properties in every bird which we create. Instead, we can simply *inherit* the bird class in all the birds which we create.

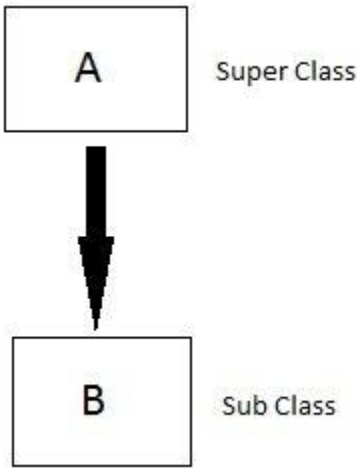
We have 5 different types of Inheritance. Namely,

1. Single Inheritance
2. Multiple Inheritance

3. Hierarchical Inheritance
4. Multilevel Inheritance
5. Hybrid Inheritance (also known as Virtual Inheritance)

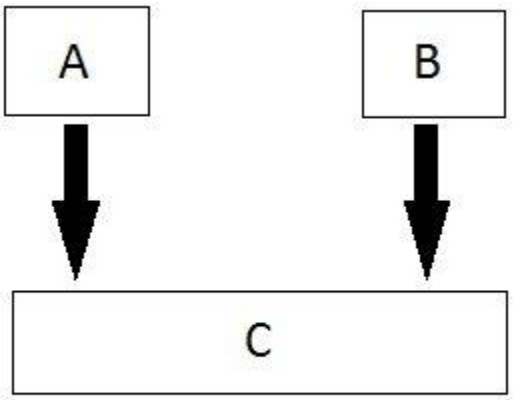
Single Inheritance in C++

In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance.



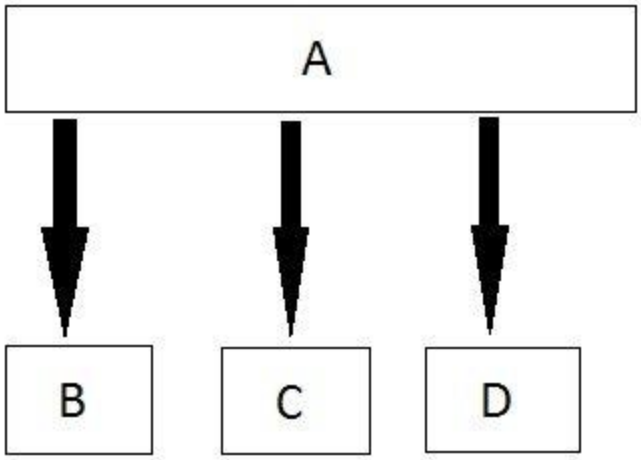
Multiple Inheritance in C++

In this type of inheritance a single derived class may inherit from two or more than two base classes.



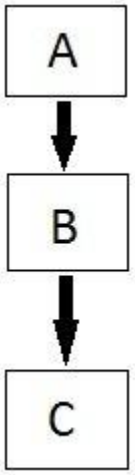
Hierarchical Inheritance in C++

In this type of inheritance, multiple derived classes inherit from a single base class.



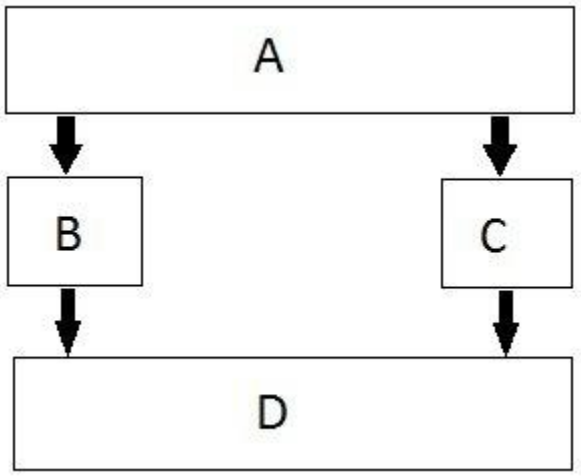
Multilevel Inheritance in C++

In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is a sub class for the other.



Hybrid (Virtual) Inheritance in C++

Hybrid Inheritance is a combination of Hierarchical and Multilevel Inheritance.



Pillar4:Polymorphism

Polymorphism comes from the Greek words “poly” and “morphism”. “poly” means many and “morphism” means form i.e.. many forms. Polymorphism means the ability to take more than one form. For example, an operation has different behavior

in different instances. The behavior depends upon the type of the data used in the operation.

Let us look at the example of a car. A car has a gear transmission system. It has four front gears and one backward gear. When the engine is accelerated then depending upon which gear is engaged different amounts of power and movement is delivered to the car. The action is same applying gear but based on the type of gear the action behaves differently or you can say that it shows many forms (polymorphism means many forms)

What are the benefits of OOP?

Benefits of OOP include:

- **Modularity.** Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.
- **Reusability.** Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.

- **Productivity.** Programmers can construct new programs quicker through the use of multiple libraries and reusable code.
- **Easily upgradable and scalable.** Programmers can implement system functionalities independently.
- **Interface descriptions.** Descriptions of external systems are simple, due to message passing techniques that are used for object communication.
- **Security.** Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and internet protocols are protected.
- **Flexibility.** Polymorphism enables a single function to adapt to the class it is placed in. Different objects can also pass through the same interface.

Advantages

Below are the advantages:

- A real-world idea can be demonstrated, as everything in OOP is treated as an object.
- As we use the concept of encapsulation, programs are easier to test and maintain.
- Faster development of code is done as we develop classes parallel instead of sequentially.
- OOP provides greater security due to data abstraction. The outside world cannot access the hidden data.
- Reusability can be achieved by using classes that have been already written.

Disadvantages

Below are the disadvantages:

- Designing a program with an OOP concept can be tricky.
- A programmer needs to plan beforehand for developing a program in OOP.
- The size of programs developed with OOP is bigger than those developed with a procedural approach.

- Since OOP programs are larger in size, the execution time for these programs is also more.