# Object-Oriented Programming

**Name: Yutika Prabhu**

Object-Oriented Programming is known as OOPs.

In object-oriented programming, objects that include both data and functions are created, whereas in procedural programming, procedures or functions that perform actions on the data are written.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Real-world concepts like inheritance, hiding, polymorphism, etc., are intended to be implemented in programming through object-oriented programming. OOP's primary goal is to bind together the data and the functions that work with it, preventing access to the data by any other portion of the code except that function.

There are some basic concepts that act as the building blocks of OOPs i.e.

- Class
- Objects
- Encapsulation
- Abstraction
- Polymorphism
- Inheritance
- Dynamic Binding
- Message Passing

**C++ Classes/Objects:**

C++ is an object-oriented programming language.

In C++, everything is connected to classes and objects, as well as their methods and attributes. For instance, an automobile is an object in the actual world. The car contains features like color and weight, as well as functions like brake and drive.

In essence, variables and functions that are a part of the class are attributes and methods. These are called "class members" a lot.

A class serves as an object constructor, or a "blueprint" for producing, and is a user-defined data type that may be used in programs.

**Class**: It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

**Example**: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, the Car is the class, and wheels, speed limits, and mileage are their properties.

```
class MyClass {      // The class
  public:            // Access specifier
    int myNum;       // Attribute
    string myString;  // Attribute
};
```

**Object:**

An object in C++ is a real-world object, such as a chair, automobile, pen, smartphone, laptop, etc. Stated differently, an object is an entity with both state and behavior. State in this context refers to data, and behavior to functioning. An object is a creation that occurs during runtime.

**Class Methods:**

Methods are functions that belongs to the class.

There are two ways to define functions that belongs to a class:

- Inside class definition
- Outside class definition

In the following example, we define a function inside the class, and we name it "myMethod".

**Encapsulation**

Encapsulation is defined as binding together the data and the functions that manipulate them.

Example:

Consider, in a company, there are different sections like the accounts section, finance section, sales section, etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is.

**Abstraction:**

Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Example:

a man driving a car. The man only knows that pressing the accelerator will increase the speed of the car or applying brakes will stop the car but he does not know how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of an accelerator, brakes, etc. in the car.

**Polymorphism:**

The definition of polymorphism is having multiple forms. The capacity of a message to appear in multiple Forms is known as polymorphism.

There are two main types of polymorphism: compile-time polymorphism, also known as method overloading, and runtime polymorphism, also known as method overriding.

Example:

A person at the same time can have different characteristics. A man at the same time is a father, a husband, and an employee. So the same person possesses different behaviors in different situations.

C++ supports operator overloading and function overloading.

1. Operator Overloading: The process of making an operator exhibit

different behaviors in different instances is known as operator

overloading.

2. Function Overloading: Function overloading is using a single function

name to perform different types of tasks. Polymorphism is extensively

used in implementing inheritance.

Different approaches to Operator Overloading

You can perform operator overloading by implementing any of the following types of functions:

- Member Function
- Non-Member Function
- Friend Function

Restrictions to Operator Overloading

- The operators = (scope resolution). (member access). (member access through pointer to member), and? (ternary conditional) cannot be overloaded.
- It is not possible to change the precedence, grouping, or number of operands operators
- The overload of operator must either return a raw pointer, or return an object (by reference or by value) for which operator is in turn overloaded

- The overloads of operators & and lose stron-Circuit Evaluation

**Inheritance:**

The capability of a class to derive properties and characteristics from another class is called Inheritance.

Example: Dog, Cat, Cow can be Derived Class of Animal Base Class.

Types Of Inheritance:-

- Single inheritance
- Multilevel inheritance
- Multiple inheritance
- Hierarchical inheritance
- Hybrid inheritance

**Dynamic Binding:**

In dynamic binding, the code to be executed in response to the function call is decided at runtime.

**Message Passing:**

Objects communicate with one another by sending and receiving information. A message for an object is a request for the execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results.

Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

**Access Specifiers:**

In C++, there are three access specifiers:

- public - members are accessible from outside the class
- private - members cannot be accessed (or viewed) from outside the class

- protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes. You will learn more about Inheritance later.

## Default Copying

By default, objects can be copied. In particular, a class object can be initialized with a copy of an object of its class. For example:

Date d1 = my_birthday; // initialization by copy

Date d2 {my_birthday}; // initialization by copy

By default, the copy of a class object is a copy of each member.

If that default is not the behavior wanted for a class X, a more appropriate behavior can be provided using overloading.

## Static Members of a C++ Class

We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

## Passing and Returning Objects in C++

In C++ we can pass class objects as arguments and also return them from a function the same way we pass and return other variables.

## Passing an Object as argument

To pass an object as an argument we write the object name as the argument while calling the function the same way we do it for other variables.

**Syntax:**

fun(object_name);

## Friend Class and Function in C++:

A friend class can access private and protected members of other classes in which it is declared as a friend. It is sometimes useful to allow a particular class to access private and protected members of other classes. For example, a LinkedList class may be allowed to access private members of Node. A friend class can access private and protected members of other class which it is declared as friend.

We can declare a friend class in C++ by using the friend keyword.

Syntax:

```
friend class class_name;    // declared in the base class
```

**Pointers**

Pointers are symbolic representations of addresses. They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures. Iterating over elements in arrays or other data structures is one of the main use of pointers.

The address of the variable you're working with is assigned to the pointer variable that points to the same data type (such as an int or string).

Syntax:

```
datatype *var_name;

int *ptr;
```