Object Oriented Programming in C++

Before Object-Oriented Programming (OOPs), most programs used a procedural approach, where the focus was on writing step-by-step functions. This made it harder to manage and reuse code in large applications.

OOP in C++ was introduced to solve this problem by organizing code into classes and objects, making programs easier to understand, reuse, and maintain.

Structures code into logical units (classes and objects)

Keeps related data and methods together (encapsulation)

Makes code modular, reusable and scalable

Prevents unauthorized access to data

Follows the DRY (Don't Repeat Yourself) principle

Class

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. Using classes, you can create multiple objects with the same behavior instead of writing their code multiple times. In general, class declarations in C++ can include these components.

Access Specifiers: A class can have members defined as public, private, or protected to control accessibility.

Class Name: The class name should follow naming conventions, usually starting with a capital letter.

Body: The class body is enclosed with braces {} and defines data members and member functions.

```
#include <iostream>
using namespace std;
class Student {
public:
   string name;
   int age;
   void display() {
      cout << name << endl;
   }
};
```

```
int main() {
   // Create object
   Student s1;

   // Assign values
   s1.name = "Geeksforgeeks";

   // Call function
   s1.display();

   return 0;
}
```

Output
Geeksforgeeks

Object

An Object is a basic unit of Object-Oriented Programming that represents real-life entities. A typical C++ program creates many objects, which interact with each other by invoking methods. The objects are what perform your code, they are the part of your code visible to the user. An object mainly consists of:

State: It is represented by the data members (attributes) of an object. It also reflects the properties of an object.

Member Function: A member function is a collection of statements that perform some specific task and may return the result to the caller.

Behavior: It is represented by the member functions of an object. It also reflects the response of an object to other objects.

Identity: It is a unique name or reference given to an object that enables it to interact with other objects.

```
#include <iostream>
#include <string>
using namespace std;

class Employee {

   // Instance variables
```

```
private:
    string name;
    float salary;


public:
    // Constructor
    Employee(string name, float salary) {
        this->name = name;
        this->salary = salary;
    }

    // getters method
    string getName() { return name; }
    float getSalary() { return salary; }

    // setters method
    void setName(string name) { this->name = name; }
    void setSalary(float salary) { this->salary = salary; }

    // Instance method
    void displayDetails() {
        cout << "Employee: " << name << endl;
        cout << "Salary: " << salary << endl;
    }
};


int main() {
    Employee emp("Geek", 10000.0f);
    emp.displayDetails();
    return 0;
}
```

Output

Employee: Geek

Salary: 10000

Four Pillars of OOP in C++:

1. Abstraction

Abstraction in C++ is the process of hiding the implementation details and only showing the essential details or features to the user. It allows to focus on what an object does rather than how it does it. In C++ abstraction is achieved using abstract classes (classes that have at least one pure virtual function).

2. Encapsulation

Encapsulation is the process of bundling data and methods into a single unit (class) and restricting direct access to some of its components. It acts as a protective shield for the data.

Data is hidden from other classes and accessed only through public methods.

Achieved by declaring class variables as private and providing public getter/setter methods.

Ensures data integrity and controlled access.

Encapsulation in C++

3. Inheritance

Inheritance is a mechanism in C++ where a class (derived) acquires the properties and behaviors of another class (base), forming an "is-a" relationship.

Enables code reusability by inheriting data members and methods from the base class.

Achieved using : followed by an access specifier (public, private, protected).

Example: Dog, Cat, Cow can be derived classes of the Animal base class.

4. Polymorphism

The word polymorphism means having many forms, and it comes from the Greek words poly (many) and morph (forms), this means one entity can take many forms. In C++, polymorphism allows the same method or object to behave differently based on the context, specially on the project's actual runtime class.

Advantage of OOP over Procedure-Oriented Programming Language.

By using objects and classes, you can create reusable components, leading to less duplication and more efficient development.

It provides a clear and logical structure, making the code easier to understand, maintain, and debug.

OOP support the DRY (Don't Repeat Yourself) principle. This principle encourages minimizing code repetition, leading to cleaner, more maintainable code.

By reusing existing code and creating modular components, OOP allows for quicker and more efficient application development.

Disadvantage of OOP

OOP has concepts like classes, objects, inheritance etc. For beginners, this can be confusing and takes time to learn.

If we write a small program, using OOP can feel too heavy. We might have to write more code than needed just to follow the OOP structure.

The code is divided into different classes and layers, so in this, finding and fixing bugs can sometimes take more time.

OOP creates a lot of objects, so it can use more memory compared to simple programs written in a procedural way.