C++ OOPs Concepts

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.

Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism etc.**

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. **Smalltalk** is considered as the first truly object-oriented programming language.

## OOPs (Object Oriented Programming System)

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:



* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

### Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

### Class

**Collection of objects** is called class. It is a logical entity.

### Inheritance

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

### Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In C++, we use Function overloading and Function overriding to achieve polymorphism.

### Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In C++, we use abstract class and interface to achieve abstraction.

### Encapsulation

**Binding (or wrapping) code and data together into a single unit is known as encapsulation.** For example: capsule, it is wrapped with different medicines.

## Advantage of OOPs over Procedure-oriented programming language

1. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
2. OOPs provide data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
3. OOPs provide ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

# C++ Object and Class

Since C++ is an object-oriented language, program is designed using objects and classes in C++.

## C++ Object

In C++, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.

In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.

Object is a runtime entity, it is created at runtime.

Object is an instance of a class. All the members of the class can be accessed through object.

Let's see an example to create object of student class using s1 as the reference variable.

Student s1;  //creating an object of Student

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class.

## C++ Class

In C++, object is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

1. **class** Student
2. {
3. **public**:
4. **int** id;  //field or data member
5. **float** salary; //field or data member
6. String name;//field or data member
7. }

## C++ Object and Class Example

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Student {
4. **public**:
5. **int** id;//data member (also instance variable)
6. string name;//data member(also instance variable)
7. };
8. **int** main() {
9. Student s1; //creating an object of Student
10. s1.id = 201;
11. s1.name = "Sonoo Jaiswal";
12. cout<<s1.id<<endl;
13. cout<<s1.name<<endl;
14. **return** 0;
15. }

Output:

201

Sonoo Jaiswal

## C++ Class Example: Initialize and Display data through method

Let's see another example of C++ class where we are initializing and displaying object through method.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Student {
4. **public**:
5. **int** id;//data member (also instance variable)
6. string name;//data member(also instance variable)
7. **void** insert(**int** i, string n)
8. {
9. id = i;
10. name = n;
11. }
12. **void** display()
13. {
14. cout<<id<<"  "<<name<<endl;
15. }
16. };
17. **int** main(**void**) {
18. Student s1; //creating an object of Student
19. Student s2; //creating an object of Student
20. s1.insert(201, "Sonoo");
21. s2.insert(202, "Nakul");
22. s1.display();
23. s2.display();
24. **return** 0;
25. }

Output:

201 Sonoo

202 Nakul

## C++ Class Example: Store and Display Employee Information

Let's see another example of C++ class where we are storing and displaying employee information using method.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Employee {
4. **public**:
5. **int** id;//data member (also instance variable)
6. string name;//data member(also instance variable)
7. **float** salary;
8. **void** insert(**int** i, string n, **float** s)
9. {
10. id = i;
11. name = n;
12. salary = s;
13. }
14. **void** display()
15. {
16. cout<<id<<"  "<<name<<"  "<<salary<<endl;
17. }
18. };
19. **int** main(**void**) {
20. Employee e1; //creating an object of Employee
21. Employee e2; //creating an object of Employee
22. e1.insert(201, "Sonoo",990000);
23. e2.insert(202, "Nakul", 29000);
24. e1.display();
25. e2.display();
26. **return** 0;
27. }

Output:

201 Sonoo 990000

202 Nakul 29000