Object Oriented Programming in C++

Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data 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

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.

For 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 {
  public:
    int myNum;
    string myString;
};

int main() {
  MyClass myObj;
  myObj.myNum = 15;
  myObj.myString = "Hi";
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Objects:

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.

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 have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is.

```
// Program to calculate the area of a rectangle
#include <iostream>
using namespace std;
class Rectangle {
  public
    int length;
    int breadth;
    Rectangle(int len, int brth) : length(len), breadth(brth) {}
    int getArea() {
      return length * breadth;
    }
};
int main() {
  Rectangle rect(8, 6);
  cout << "Area = " << rect.getArea();

  return 0;
}
```

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 actually increasing, he does not know about the inner mechanism of the car or the implementation of an accelerator, brakes, etc. in the car.

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

class implementAbstraction {
private:
    int a, b;

public:
    void set(int x, int y)
    {
        a = x;
        b = y;
    }

    void display()
    {
        cout << "a = " << a << endl;
        cout << "b = " << b << endl;
    }
};

int main()
{
    implementAbstraction obj;
    obj.set(10, 20);
    obj.display();
    return 0;
}
```

Polymorphism:

The word polymorphism means having many forms.we can define polymorphism as the ability of a message to be displayed in more than one form.

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 behavior in different situations.

C++ supports operator overloading and function overloading.

- Operator Overloading: The process of making an operator exhibit different behaviors in different instances is known as operator overloading.
- Function Overloading: Function overloading is using a single function name to perform different types of tasks. Polymorphism is extensively used in implementing inheritance.

Function overloading

```
#include <bits/stdc++.h>

using namespace std;
class Value {
public:

    void func(int x)
    {
        cout << "value of x is " << x << endl;
    }
    void func(double x)
    {
        cout << "value of x is " << x << endl;
    }

    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y
             << endl;
    }
};
```

```
int main()
{
    Value obj1;
    obj1.func(7);
    obj1.func(9.132);
    obj1.func(85, 64);
    return 0;
}
```

Operator Overloading:

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

class Complex {
private:
    int real, imag;

public:
    Complex(int r = 0, int i = 0)
    {
        real = r;
        imag = i;
    }
    Complex operator+(Complex const& obj)
    {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }
    void print() { cout << real << " + i" << imag << endl; }
};

int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2;
    c3.print();
}
```

Function overriding

```
#include <bits/stdc++.h>
using namespace std;

class Animal {
public:
   string color = "Black";
};

class Dog : public Animal {
public:
   string color = "Grey";
};
int main(void)
{
   Animal d = Dog();
   cout << d.color;
}
```

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.

```
class Vehicle {
 public:
   string brand = "Ford";
   void honk() {
     cout << "Tuut, tuut! \n" ;
   }
};
class Car: public Vehicle {
 public:
   string model = "Mustang";
};
int main() {
 Car myCar;
 myCar.honk();
 cout << myCar.brand + " " + myCar.model;
 return 0;
}
```

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.

```
#include <iostream>
using namespace std;
class Trial {
public:
   void call_Function()
   {
      print();
   }
   void print()
   {
      cout << "Printing the Base class Content" << endl;
   }
};
class Trial2 : public Trial
{
public:
   void print()
   {
      cout << "Printing the Derived class Content"
          << endl;
   }
};
int main()
{
   Trial try;
   try.call_Function();
   Trial2 try2;
   try.call_Function();
   return 0;
}
```

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:

Access specifiers define how the members (attributes and methods) of a class can be accessed.

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.

Constructor:

Constructor in C++ is a special method that is invoked 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 the class or structure.

```
#include<iostream>
using namespace std;
class student
{
    int rno;
    char name[50];
    double fee;
    public:
    student()
    {
        cout<<"Enter the RollNo:";
```

```
         cin>>rno;
        cout<<"Enter the Name:";
        cin>>name;
        cout<<"Enter the Fee:";
        cin>>fee;
    }
    void display()
    {
        cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;
    }
};
int main()
{
    student s;
    s.display();
    return 0;
}
```