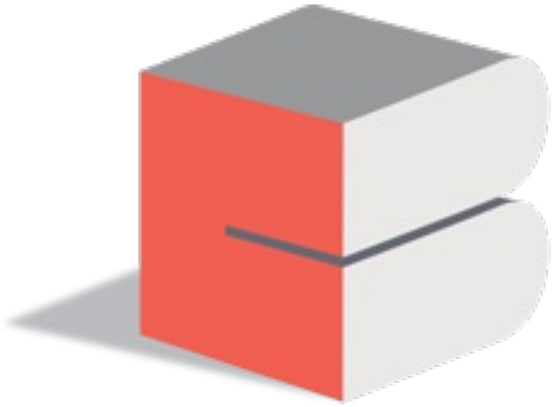

Object Oriented Programming - It’s Features

Inheritance

- Capability of a class to derive properties and characteristics from another class.
- **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class:** The class whose properties are inherited by sub class is called Base Class or Super class.

Why and when to use inheritance?

Example Consider a group of vehicles. You need to create classes for Bus, Car and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be same for all of the three classes. If we create these classes avoiding inheritance then we have to write all of these functions in each of the three classes.

To avoid the chances of error and data redundancy, inheritance is used.

Using inheritance, we have to write the functions only one time instead of three times as we have inherited rest of the three classes from base class(Vehicle).

```
class subclass_name : access_mode base_class_name
{
  //body of subclass
};
```

Example Code

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

//Base class
class Parent
{
    public:
      int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
    public:
```

```
      int id_c;
};

//main function
int main()
{

    Child obj1;

    // An object of class child has all data members
    // and member functions of class parent
    obj1.id_c = 7;
    obj1.id_p = 91;
    cout << "Child id is " <<  obj1.id_c << endl;
    cout << "Parent id is " <<  obj1.id_p << endl;

    return 0;
}
```

OUTPUT

```
Child id is 7
Parent id is 91
```

Modes of Inheritance

- **Public mode:** If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class. Private members of the base class will never get inherited in sub class.

- **Protected mode:** If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class. Private members of the base class will never get inherited in sub class.

- **Private mode:** If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class. Private members of the base class will never get inherited in sub class.

Types of Inheritance in C++

1. **Single Inheritance:** A class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.

```
class subclass_name : access_mode base_class
{
  //body of subclass
};
```

2. **Multiple Inheritance:** Multiple Inheritance is a feature of C++

where a class can inherit from more than one classes. i.e one sub class is inherited from more than one base classes.

```
class subclass_name : access_mode base_class1, access_mode ba
se_class2, ....
{
  //body of subclass
};
```

3. **Multilevel Inheritance:** In this type of inheritance, a derived class is created from another derived class.

4. **Hierarchical Inheritance:** In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single base class.

5. **Hybrid (Virtual) Inheritance:** Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

Polymorphism

- polymorphism means having many forms.
- the ability of a message to be displayed in more than one form.

In C++ polymorphism is mainly divided into two types:

1. Compile time polymorphism: This type of polymorphism is achieved by function overloading or operator overloading. (explained in OOPs III Notes)
2. Runtime polymorphism: This type of polymorphism is achieved by Function Overriding.

- Function overriding on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

EXAMPLE: FUNCTION OVERRIDING

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

// Base class
class Parent
{
    public:
    void print()
    {
        cout << "The Parent print function was called" << end
l;
    }
};

// Derived class
class Child : public Parent
```

```
{
    public:

    // definition of a member function already present in Par
ent
    void print()
    {
        cout << "The child print function was called" << endl
;
    }

};

//main function
int main()
{
    //object of parent class
    Parent obj1;

    //object of child class
    Child obj2 = Child();


    // obj1 will call the print function in Parent
    obj1.print();

    // obj2 will override the print function in Parent
    // and call the print function in Child
```

```
    obj2.print();
    return 0;
}
```

OUTPUT

```
The Parent print function was called
The child print function was called
```

Encapsulation

- binding together the data and the functions that manipulates them.
- Encapsulation also lead to data abstraction or hiding.

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

class Encapsulation
{
    private:
        // data hidden from outside world
        int x;

    public:
        // function to set value of
        // variable x
        void set(int a)
```

```
        {
            x =a;
        }

        // function to return value of
        // variable x
        int get()
        {
            return x;
        }
};

// main function
int main()
{
    Encapsulation obj;

    obj.set(5);

    cout<<obj.get();
    return 0;
}
```

Note The variable x is made private. This variable can be accessed and manipulated only using the functions get() and set() which are present inside the class. Thus we can say that here, the variable x and the functions get() and set() are binded together which is nothing but encapsulation.

Role of access specifiers in encapsulation

As we have seen in above example, access specifiers plays an important role in implementing encapsulation in C++. The process of implementing encapsulation can be sub-divided into two steps:

- The data members should be labeled as private using the private access specifiers.
- The member function which manipulates the data members should be labeled as public using the public access specifier.

Abstraction

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

Access specifiers are the main pillar of implementing abstraction in C++. We can use access specifiers to enforce restrictions on class members. For example:

- Members declared as public in a class, can be accessed from anywhere in the program.
- Members declared as private in a class, can be accessed only from

within the class. They are not allowed to be accessed from any part of code outside the class.

Example

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

class implementAbstraction
{
    private:
        int a, b;

    public:

        // method to set values of
        // private members
        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;
}
```

OUTPUT

```
a = 10
b = 20
```

You can see in the above program we are not allowed to access the variables a and b directly, however one can call the function set() to set the values in a and b and the function display() to display the values of a and b.

Advantages

- Helps the user to avoid writing the low level code
- Avoids code duplication and increases reusability.
- Can change internal implementation of class independently without affecting the user.
- Helps to increase security of an application or program as only important details are provided to the user.

Dynamic Binding

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

Message Passing

Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for 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.