Object Oriented Programming in C++

Class

Objects

Inheritance

Encapsulation

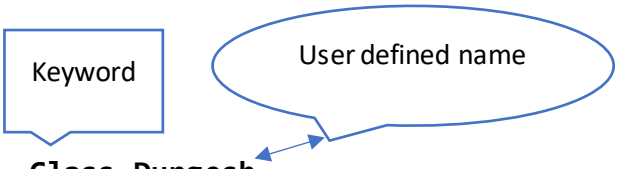
Abstraction

Polymorphism

Class

class is the building block, that leads to Object-Oriented programming. 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(object) of that class.

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.




```
Class Durgesh
{ // class start with curly brackets
      public:
     int id;// field or data member
float salary;// field or data member
String name;// field or data member
void insert()    // function or method
  {              }
 void display()    // function or method
{           }
}; // class name ends with a semicolon.
```

Class is a user defined datatype which has data members and member functions.

Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

Objects:

An Object is an instance(object) of a class. When a class is defined, no memory is allocated but when an object is created memory is allocated for object.

To use the data and access functions defined in the class, we need to create an object.

Defining Class and Declaring Objects. A class is defined in C++ using keyword class followed by the name of class.

Syntax:

```
ClassName ObjectName;

Durgesh    d;//creating an object of Durgesh

Class Durgesh

{

      int id;// field or data member

float salary;// field or data member

String name;// field or data member

};
```

C++ Object and Class Example

```
#include <iostream>
using namespace std;
class Student {
  public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable)
};
int main()
{
   Student s1; //creating an object of Student
   s1.id = 3021;
   s1.name = "Durgesh";
   cout<<s1.id<<endl;
   cout<<s1.name<<endl;
   return 0;
}
```

Note

1. The. (dot) operator are used to reference individual members of classes, structures, and unions.
2. The dot operator is applied to the actual object.
3. The arrow operator is used with a pointer to an object.

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

class Car {
  public:
    string brand;
    string model;
    int year;
};

int main() {
  Car carObj1;
  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;

  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;

  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;
}
```

C++ Class Example: Initialize and Display data through function or method for practice

(1)

```
#include <iostream>
using namespace std;
class Student {
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
     void insert(int i, string n)
     {
        id = i;
        name = n;
     }
     void display()
     {
        cout<<id<<"  "<<name<<endl;
     }
};
int main(void){
   Student s1; //creating an object of Student
   Student s2; //creating an object of Student
   s1.insert(201, "Durgesh");
   s2.insert(202, "sushant");
   s1.display();
   s2.display();
   return 0;
}
```

(2) same program with little change

```
#include <iostream>
using namespace std;
class Student {
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
     void insert(int i, string n)
     {
        id = i;
        name = n;
        cout<<id<<"  "<<name<<endl;
     }

};
int main(void) {
   Student s1; //creating an object of Student
   Student s2; //creating an object of Student
   s1.insert(201, "Durgesh");
   s2.insert(202, "sushant");

   return 0;
}
```

C++ Inheritance

The process of getting properties and behaviors from one class to another class is called inheritance.

Properties: variables

Behaviors: methods

1. The main purpose of the inheritance is to automatically the code is reused.

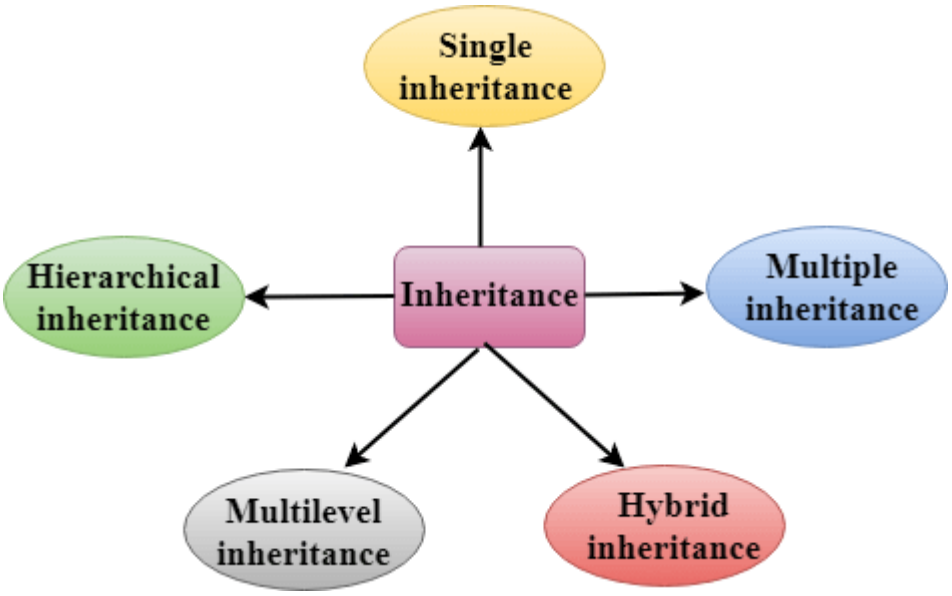
2. By using **: public** we are achieving inheritance concept in C++.

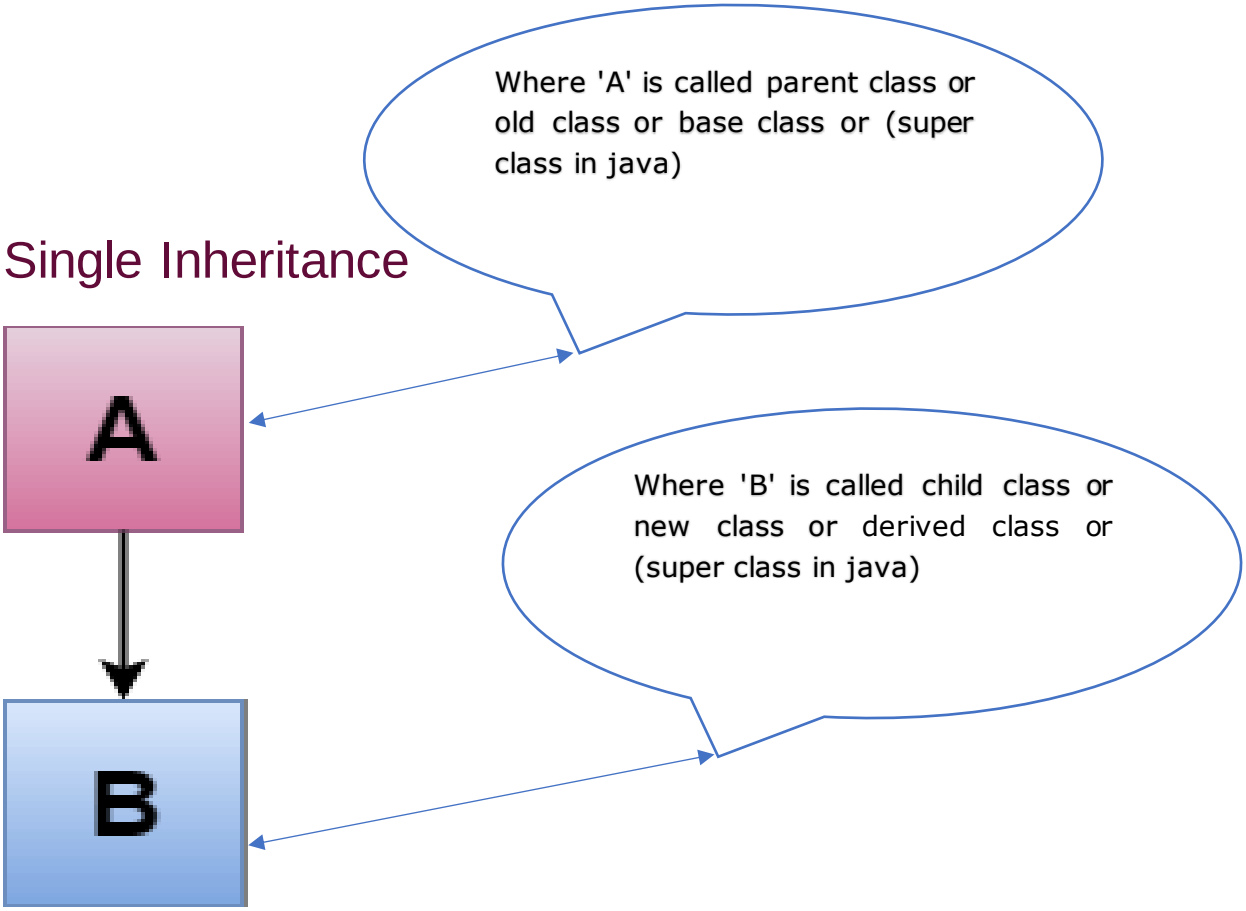
3. In the inheritance the person who is giving the properties is called parent the person who is taking the properties is called child.

4. we used inheritance to reduce length of the code.

Types of Inheritance

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



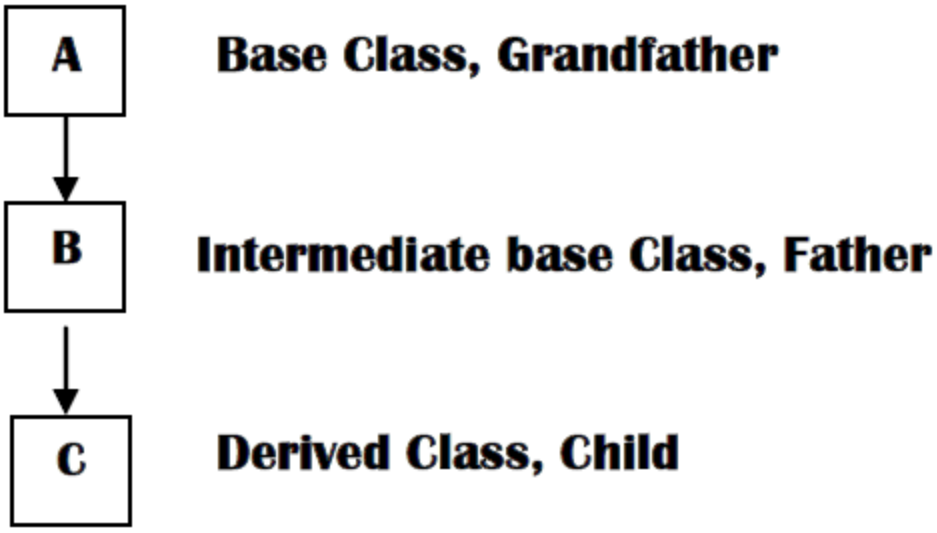



```
#include <iostream>
using namespace std;
 class Fruit {
   public:
 void D1() {
    cout<<"I am parent..."<<endl;
 }
  };
  class Mango: public Fruit
  {
      public:
     void D2()
```

```
    {
    cout<<"I am child...";
    }
  };
int main(void) {
    Mango m1;
    m1.D1();
    //m1.D1();
    //m1.D1();
    m1.D2();
    return 0;
}
```

Multilevel Inheritance

Multilevel inheritance is a process of creating a child class from another child class.



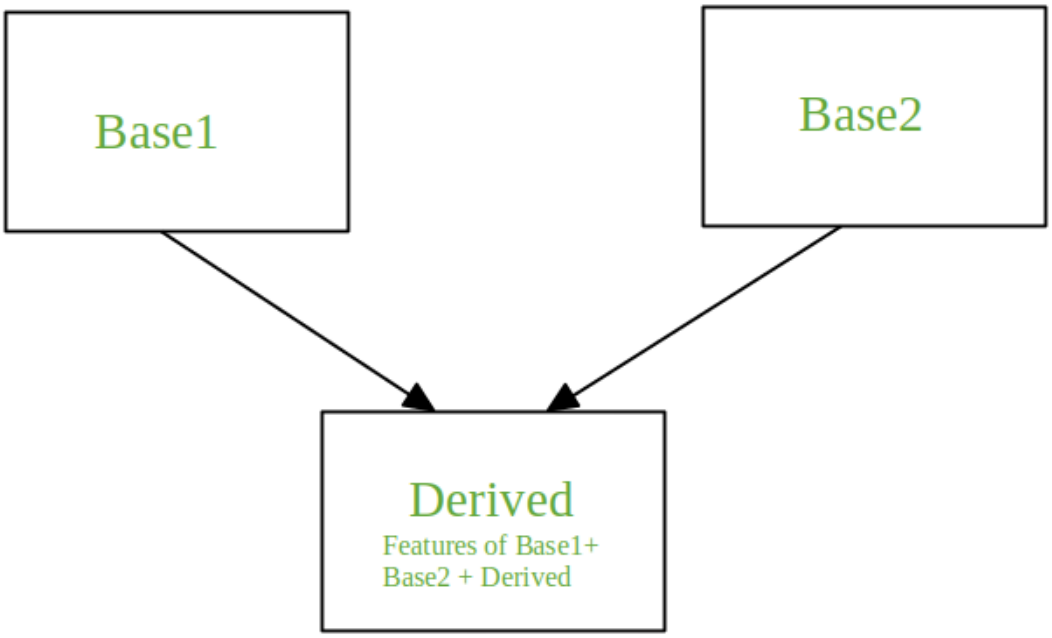
```
#include <iostream>
using namespace std;
 class Animal {
  public:
 void eat() {
   cout<<"Eating..."<<endl;
 }
  };
  class Dog: public Animal
  {
     public:
    void bark(){
   cout<<"Barking..."<<endl;
    }
  };
  class BabyDog: public Dog
  {
     public:
    void weep() {
   cout<<"Weeping...";
    }
  };
int main(void) {
   BabyDog d1;
   d1.eat();
   d1.bark();
    d1.weep();
```

```
    return 0;
}
```

Output:

```
Eating...
Barking...
Weeping...
```

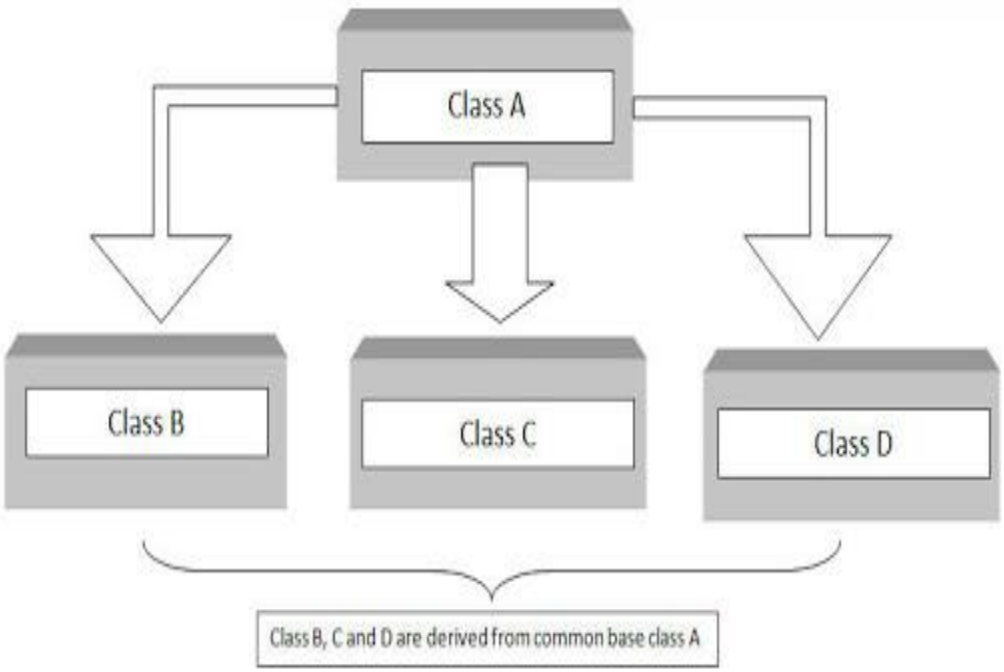
C++ Multiple Inheritance



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

class A
{
public:
 void Shankar()
{
cout <<"I am Shankar\n";

}
};

class B
{
public:
 void Pravati()
 {
cout << "I am Pravati\n";
}
};

class C: public B, public A  //  order is importanat to know tha parent
{
public:
  void Jaalandhar()
{
cout << "I am Jaalandhar and my parent name is Shankar and Pravati......";
}
};

int main()
```

```
{
    C c;
    c.Shankar();
    c.Pravati();
    c.Jaalandhar();
    return 0;
}
```

C++ Hierarchical Inheritance

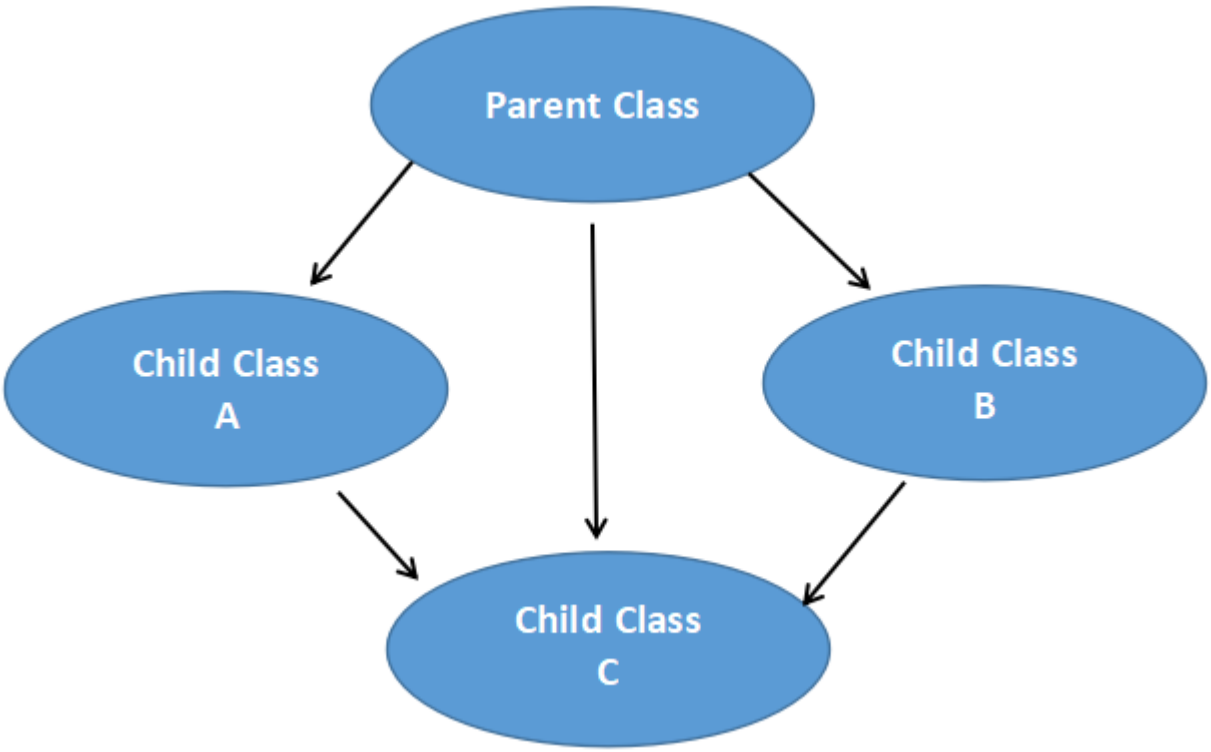


```
#include <iostream>
using namespace std;
class Shankar
{
public:

void Shankar1()
{
cout << "I am shankar,father of Karthik and Ganesh\n";
}
};
class Ganesh : public Shankar
{
public:
void Ganesh1()
{
cout << "I am Ganesh\n ";
}
};
class Karthik: public Shankar {
public:
void Karthik1()
```

```
{
cout << "I am Karthik\n ";
}
};

int main()
{
Ganesh g;

g.Shankar1();
g.Ganesh1();

Karthik k;
k.Shankar1();
k.Karthik1();
return 0;
}
```

Hybrid Inheritance

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



```
// C++ program for Hybrid Inheritance

#include <iostream>
using namespace std;


class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};

//base class
class Fare
```

```
{
   public:
   Fare()
   {
      cout<<"Fare of Vehicle\n";
   }
};

// first sub class
class Car: public Vehicle // Single level Inheritance
{

};

// second sub class
class Bus: public Vehicle, public Fare //Multiple Inheritance
{
```

```
};

// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base class
    Bus obj2;
    return 0;
}
```