Inheritance in C++

Inheritance in C++

- **Inherit Definition** - Derive quality and characteristics from parents or ancestors. Like you inherit features of your parents.
- **Example:** "She had inherited the beauty of her mother"
- Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes.
- The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.

Inheritance in C++

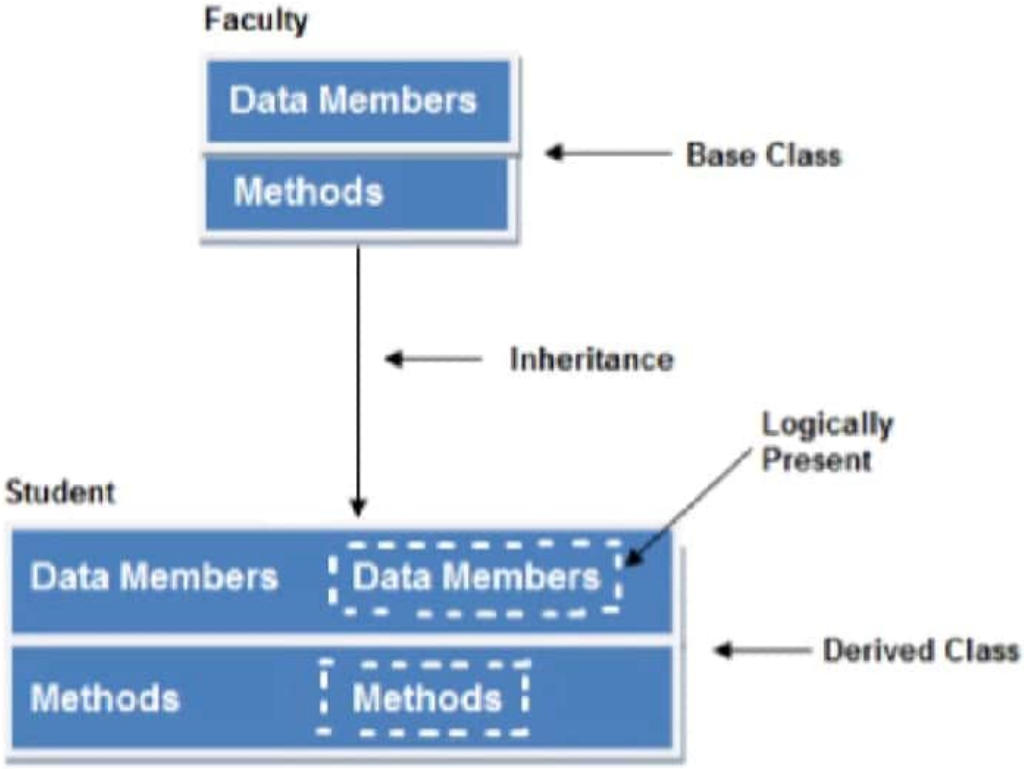
- Inheritance is the capability of one class to acquire properties and characteristics from another class. The class whose properties are inherited by other class is called the **Parent** or **Base** or **Super** class. And, the class which inherits properties of other class is called **Child** or **Derived** or **Sub** class.
- Inheritance makes the code reusable. When we inherit an existing class, all its methods and fields become available in the new class, hence code is reused.

NOTE : All members of a class except Private, are inherited

Advantage of inheritance

- If we develop any application using this concept than that application have following advantages,
- Application development time is less.
- Application take less memory.
- Application execution time is less.
- Application performance is enhance (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.
- Use of Virtual Keyword

Diagram



In the above diagram data members and methods are represented in broken line are inherited from faculty class and they are visible in student class logically.

Basic Syntax of Inheritance

```
class Subclass_name : access_mode Superclass_name
{
   // data members
   // methods
}
```

- While defining a subclass like this, the super class must be already defined or at least declared before the subclass declaration.
- Access Mode is used to specify, the mode in which the properties of superclass will be inherited into subclass, public, private or protected.
- : is operator which is used for inheriting the features of base class into derived class it improves the functionality of derived class.

Inheritance in C++ Program Example

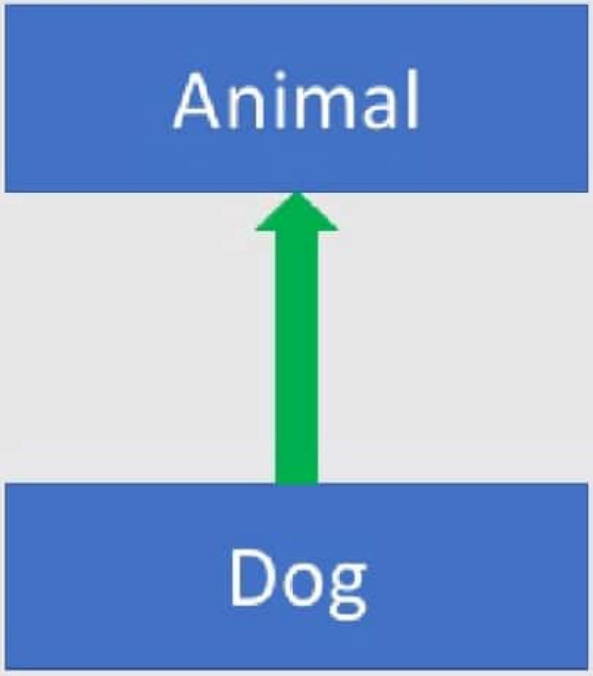
```
class Rectangle
{
        ... .. ...
};

class Area : public Rectangle
{
        ... .. ...
};

class Perimeter : public Rectangle
{
        .... .. ...
};
```

Example of Inheritance

Class Dog inherits the properties from its super class Animal



Inheritance Visibility Mode

- Depending on Access modifier used while inheritance, the availability of class members of Super class in the sub class changes. It can either be private, protected or public.

- **1) Public Inheritance**
  - This is the most used inheritance mode. In this the protected member of super class becomes protected members of sub class and public becomes public.

class Subclass : public Superclass

Inheritance Visibility Mode

- **2) Private Inheritance**
  - In private mode, the protected and public members of super class become private members of derived class.

```
class Subclass : Superclass
// By default its private inheritance
```

- **3) Protected Inheritance**
  - In protected mode, the public and protected members of Super class becomes protected members of Sub class.

```
class subclass : protected Superclass
```

Table showing all the Visibility Modes

	Derived Class	Derived Class	Derived Class
Base class	**Public Mode**	**Private Mode**	**Protected Mode**
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Access Control and Inheritance

- A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Inheritance Visibility Mode

Member Access Specifier	How Members of the Base Class Appear in the Derived Class
Private	Private members of the base class are inaccessible to the derived class.
	Protected members of the base class become private members of the derived class.
	Public members of the base class become private members of the derived class.

Inheritance Visibility Mode

Member Access Specifier	How Members of the Base Class Appear in the Derived Class
Protected	Private members of the base class are inaccessible to the derived class.
	Protected members of the base class become protected members of the derived class.
	Public members of the base class become protected members of the derived class.

Inheritance Visibility Mode

Member Access Specifier	How Members of the Base Class Appear in the Derived Class
Public	Private members of the base class are inaccessible to the derived class.
	Protected members of the base class become protected members of the derived class.
	Public members of the base class become public members of the derived class.

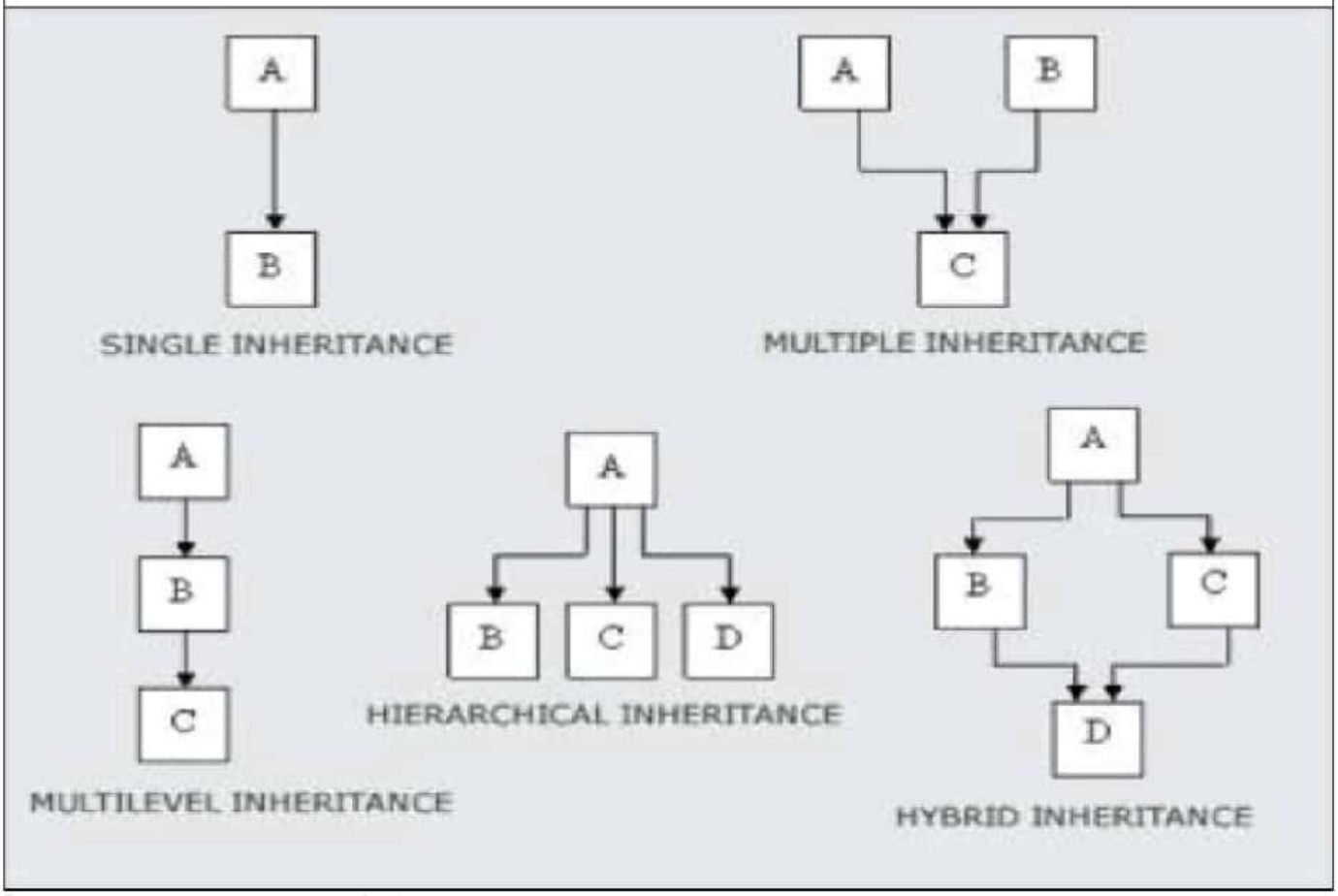
Access Control and Inheritance

- A derived class inherits all base class methods with the following exceptions:
  - Constructors, destructors and copy constructors of the base class.
  - Overloaded operators of the base class.
  - The friend functions of the base class.

Types of Inheritance

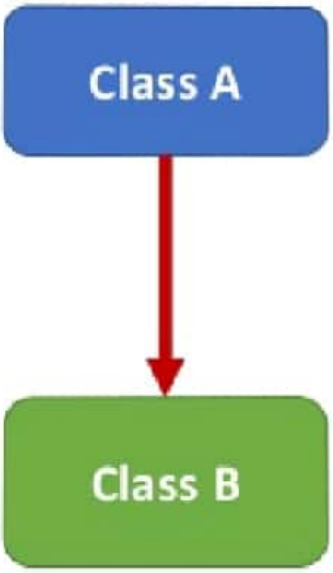
- Based on number of ways inheriting the feature of base class into derived class it have five types they are:
- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multiple inheritance
- Hybrid inheritance

Types of Inheritance



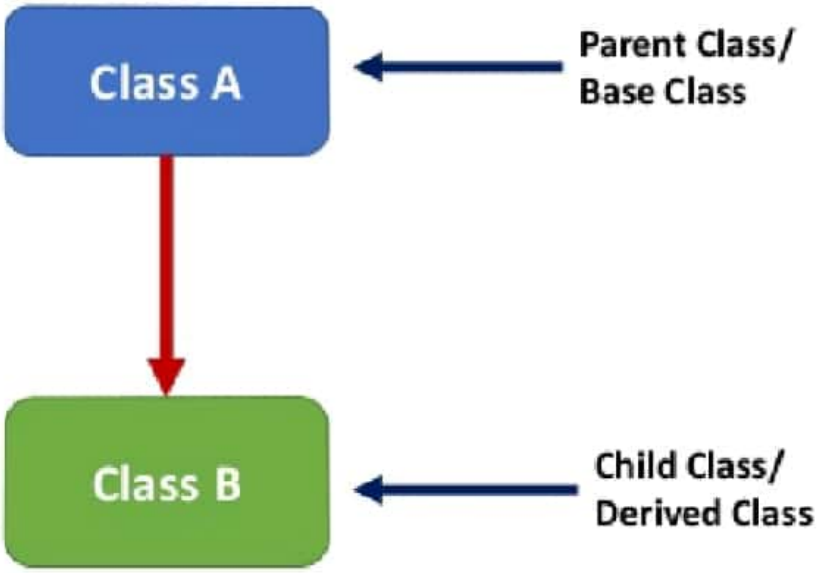
Types of Inheritance

- **Single Inheritance**
- In single inheritance there exists single base class and single derived class.
- It is the most simplest form of Inheritance.



Single Inheritance

- In single inheritance there exists single base class and single derived class.
- It is the most simplest form of Inheritance.



Single Inheritance Example

- When a single class is derived from a single parent class, it is called **Single inheritance**. It is the simplest of all inheritance.

- **For example,**
- Animal is derived from living things
- Car is derived from vehicle
- Typist is derived from staff
- Manager derived from employee
- Circle derived from shapes

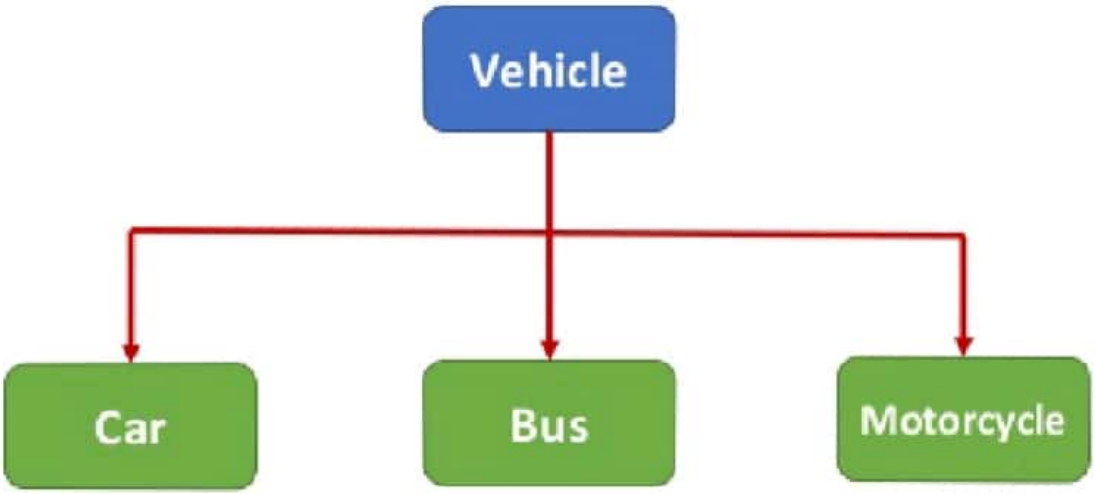
Syntax of Inheritance in C++

```
cclass base_classname
{
   properties...
   methods...
};

class derived_classname : visibility_mode base_classname
{
   properties...
   methods...
};
```

Single Inheritance Example

- **For example**: Car, Bicycle, Motorcycle are all vehicles and have many similar properties like tire, brakes, seat, etc. So they can be derived from class *Vehicle*.
- Therefore, vehicle is *base* class and car, bus, motorcycle are *derived* classes.



Single Inheritance Example

```
class A
{
        Public:
                int salary;

};


class B: public A
{


};
```

Single Inheritance Example

```
class A
{
        Public:
                int salary;

};


class B: public A
{


};
```

Class A is a Parent Class

Class B is a Child Class

Single Inheritance Example

```
class A
{
        Public:
                int salary;

};


class B: public A
{

};
```

After Inheritance Class A Member Variable goes in Class B as a Member Variable of Class B

Child class B inherit the properties of Parent class A

Example of Single Inheritance-1

```
class Person
{
        public:
        string name;
        int age;
};
class Student : public Person
{
        public:
        void show()
        {
                cout<<"Name is: "<<name<<endl;
                cout<<"Age is: "<<age;
        }
};
```

Example of Single Inheritance-1

```
class Person
{
        public:
        string name;
        int age;
};
class Student : public Person
{
        public:
        void show()
        {
                cout<<"Name is: "<<name<<endl;
                cout<<"Age is: "<<age;
        }
};
```

After Inheritance Class Student Look like this

```
class Student : public Person {
    public:
    string name;
    int age;

    public:
    void show() {
        cout<<"Name is: "<<name<<endl;
        cout<<"Age is: "<<age;
    }
};
```

Ambiguity in Single Inheritance in C++

- If parent and child classes have same named method, parent name and scope resolution operator(::) is used. This is done to distinguish the method of child and parent class since both have same name.

```
class staff {
public:
    void getdata(){}
    void display(){}
};
class typist: public staff {
public:
    void getdata(){}
    void display(){}
};
```

```
int main()
{
    typist t;
    t.display();
}
```

Above which function is called

Ambiguity in Single Inheritance in C++

- If parent and child classes have same named method, parent name and scope resolution operator(::) is used. This is done to distinguish the method of child and parent class since both have same name.

```
class staff {
public:
    void getdata(){}
    void display(){}
};
class typist: public staff {
public:
    void getdata(){}
    void display(){}
};
```

What about if we want to call staff Class Method...???

```
int main()
{
    typist t;
    t.staff::display();
}
```

Ans is use Scope Resolution Operator(::)

Example of Single Inheritance-1

In this Example we use Scope Resolution Operator(::)

```
class staff {
   private:
      char name[50];
      int code;
   public:
      void getdata();
      void display();
};
class typist: public staff {
   private:
      int speed;
   public:
      void getdata();
      void display();
};
```

Example of Single Inheritance-2

```
void staff::getdata() {
        cout<<"Name:";
        gets(name);
        cout<<"Code:";
        cin>>code;
        }
void staff::display() {
        cout<<"Name:"<<name<<endl;
        cout<<"Code:"<<code<<endl;
        }
void typist::getdata() {
        cout<<"Speed:";
        cin>>speed;
        }
void typist::display() {
        cout<<"Speed:"<<speed<<endl;
        }
```

In this Example we use Scope Resolution Operator(::)

Example of Single Inheritance-3

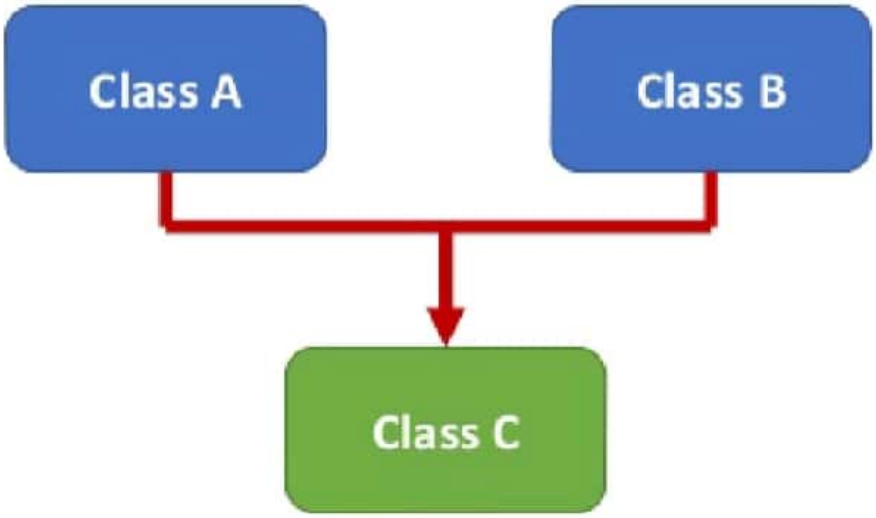
In this Example we use Scope Resolution Operator(::)

```
int main()
{
   typist t;
   cout<<"Enter data"<<endl;
   t.staff::getdata();   //calling staff class method
   t.getdata();
   cout<<endl<<"Display data"<<endl;
   t.staff::display(); //calling staff class method
   t.display();
   return 0;
}
```

In this example, typist class is derived and staff class is the base class. Public members of class staff such as staff::getdata() and staff::display() are inherited to class typist. Since the child is derived from a single parent class, it is **single inheritance**.

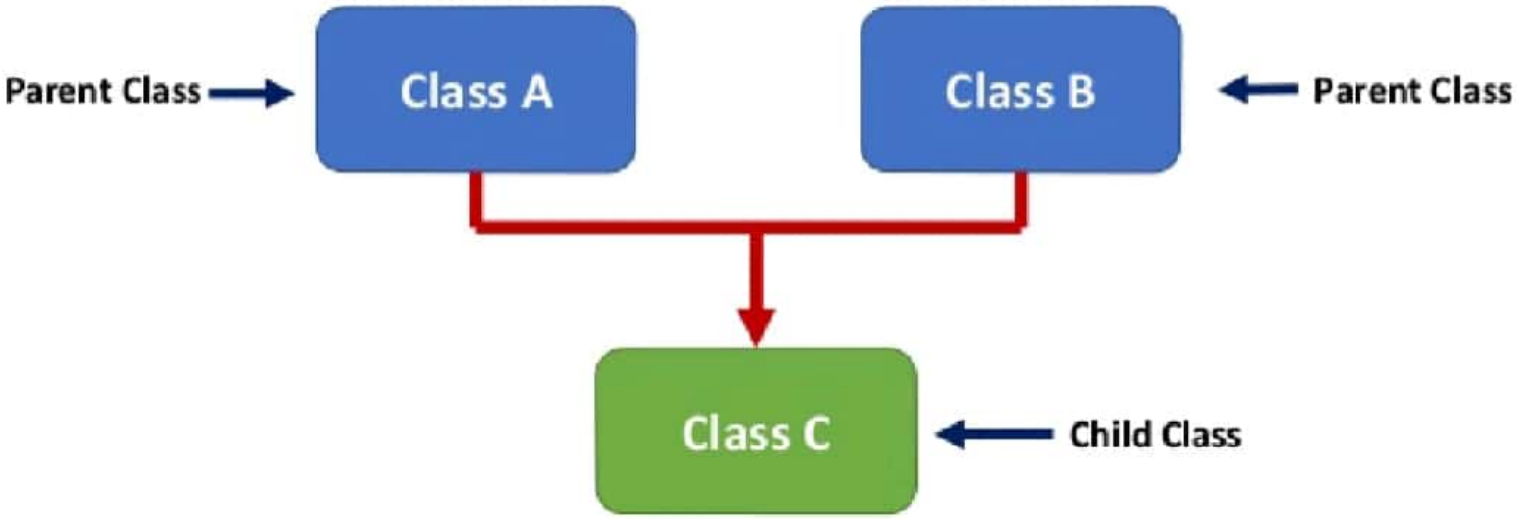
Types of Inheritance

- **Multiple Inheritance**
- In this type of inheritance a single derived class may inherit from two or more than two base classes.



Types of Inheritance

- **Multiple Inheritance**
- In this type of inheritance a single derived class may inherit from two or more than two base classes.



Multiple Inheritance Example

- When a class is derived from two or more base classes, such inheritance is called **Multiple Inheritance**. It allow us to combine the features of several existing classes into a single class.

- **For example,**
- Petrol is derived from both liquid and fuel.
- A child has character of both his/her father and mother, etc.

Syntax of Multiple Inheritance

```
class base_class1
{
   properties;
   methods;
};
class base_class2
{
   properties;
   methods;
};
class derived_classname : visibility_mode base_class1, visibility_mode base_class2
{
   properties;
   methods;
};
```

Multiple Inheritance Example

```
class A
{

};

class B
{


};


class C: public B, public A
{
};
```

Multiple Inheritance Example

```
class A
{
        public: string name;
};

class B
{
        public: int marks;
};

class C: public B, public A
{

};
```

Child Class C inherit the Properties of both Class A and Class B

After inheritance Child Class C look Like this

```
class C: public B, public A
{
    public: string name;
    public: int marks;

};
```

Ambiguity in Multiple Inheritance

- In multiple inheritance, a single class is derived from two or more parent classes. So, there may be a possibility that two or more parents have same named member function.

- If the object of child class needs to access one of the same named member function then it results in ambiguity. The compiler is confused as method of which class to call on executing the call statement.

Ambiguity in Multiple Inheritance

```
class A {
  public: void display() {
      cout <<"This is method of A";
   }
};
class B{
  public:  void display() {
      cout <<"This is method of B";
   }
};
class C: public A, public B {
  public:
};
int main() {
   C sample;
   sample.display();     /*causes ambiguity*/
}
```

Ambiguity Resolution of Multiple Inheritance

- This problem can be resolved by class name and using scope resolution operator to specify the class whose method is called.

```
derived_objectname.parent_classname::same_named_function([parameter]);
```

- In the previous example, if we want to call the method of class A then we can call it as below,

```
sample.A :: display();
```

- Similarly, if we need to call the method of class B then,

```
sample.B :: display();
```

Solution of Ambiguity in Multiple Inheritance

```
class A {
  public: void display() {
      cout <<"This is method of A";
   }
};
class B{
  public:  void display() {
      cout <<"This is method of B";
   }
};
class C: public A, public B {
  public:
};
int main() {
   C sample;
   sample.A::display();     //Now Here A Class (Parent Class) Method is Called
}
```

This is method of A

Solution of Ambiguity in Multiple Inheritance

```
class A {
  public: void display() {
      cout <<"This is method of A";
   }
};
class B{
  public:  void display() {
      cout <<"This is method of B";
   }
};
class C: public A, public B {
  public:
};
int main() {
   C sample;
   sample.B::display();     //Now Here B Class (Parent Class) Method is Called
}
```

This is method of B

One More Example of Multiple Inheritance-1

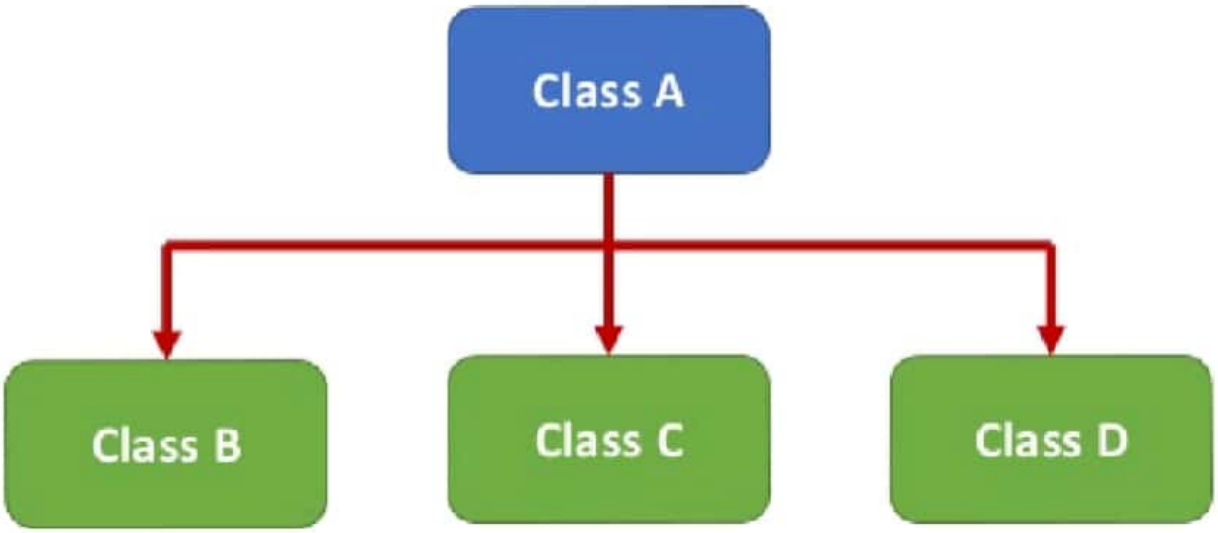
```
class student {
   protected:   int rno,m1,m2;
   public:
   void get()  {
      cout<<"Enter the Roll no :";
      cin>>rno;
      cout<<"Enter the two marks :"<<endl;
      cin>>m1>>m2;    }
};
class sports {
   protected:   int sm;  // sm = Sports mark
   public:
   void getsm() {
       cout<<"\nEnter the sports mark :"<<endl;
      cin>>sm;   }
};
```

One More Example of Multiple Inheritance-2

```
class statement:public student,public sports {
   int tot,avg;
   public:
   void display() {
      tot=(m1+m2+sm);
      avg=tot/3;
      cout<<"\n\n\tRoll No    : "<<rno<<"\n\tTotal     : "<<tot;
      cout<<"\n\tAverage    : "<<avg;
   }
};
int main() {
   statement obj;
   obj.get();
   obj.getsm();
   obj.display();
}
```

Types of Inheritance

- **Hierarchical Inheritance**
- In this type of inheritance, multiple derived classes inherits from a single base class.



Hierarchical Inheritance Example

- When more than one classes are derived from a single base class, such inheritance is known as **Hierarchical Inheritance**, where features that are common in lower level are included in parent class. Problems where hierarchy has to be maintained can be solved easily using this inheritance.

- **For Example:**
- Civil, Computer, Mechanical, Electrical are derived from Engineer.
- Natural language, Programming language are derived from Language.

Syntax of Hierarchical Inheritance

```
class base_classname {
   properties;
   methods;
};
class derived_class1:visibility_mode base_classname {
   properties;
   methods;
};
class derived_class2:visibility_mode base_classname {
   properties;
   methods;
};
... ... ...
class derived_classN:visibility_mode base_classname {
   properties;
   methods;
};
```

Hierarchical Inheritance Example

```
class A
{

};

class B: public A
{


};
class C: public A
{


};
class D: public A
{


};
```

Example of Hierarchical Inheritance-1

```
class Side {
        protected:  int l;
        public:  void set_values (int x) {
                l=x;
        }
};
class Square: public Side {
        public: int sq() {
                return (l *l);
        }
};
class Cube:public Side {
        public: int cub() {
                return (l *l*l);
        }
};
```

Side is a Parent Class

Class "Square" is Child of "Slide" Class

Class "Cube" is Child of "Slide" Class

Example of Hierarchical Inheritance-1

```
class Side {
        protected:  int l;
        public:  void set_values (int x) {
                l=x;
        }
};
class Square: public Side {
        public: int sq() {
                return (l *l);
        }
};
class Cube:public Side {
        public: int cub() {
                return (l *l*l);
        }
};
```

After Inheritance Class Square Look Like tis

```
class Square: public Side {
    protected:  int l;
    public:  void set_values (int x)
    {
        l=x;
    }

    public: int sq() {
        return (l *l);
    }
};
```

Example of Hierarchical Inheritance-1

```
class Side {
        protected:  int l;
        public:  void set_values (int x) {
                l=x;
        }
};
class Square: public Side {
        public: int sq() {
                return (l *l);
        }
};
class Cube:public Side {
        public: int cub() {
                return (l *l*l);
        }
};
```

After Inheritance Class Cube Look Like tis

```
class Cube: public Side {
    protected:  int l;
    public:  void set_values (int x)
    {
        l=x;
    }

    public: int cub() {
        return (l *l*l);
    }
};
```

Example of Hierarchical Inheritance-2

```
int main () {
        //Creating an Object of Class Square
        Square s;
        s.set_values (10);
        cout<<"--------Result---------"<<endl<<endl;
        cout << "The square value is::" << s.sq() << endl;
        //Creating an Object of Class Cube
        Cube c;
        c.set_values (20);
        cout << "The cube value is::" << c.cub() << endl;
return 0;
}
```

In the above example the two derived classes "Square", "Cube" uses a single base class "Side". Thus two classes are inherited from a single class. This is the hierarchical inheritance OOP's concept in C++.

One More Example of Hierarchical Inheritance-1

```
class Shape {
        protected:  float width, height;
        public:
        void set_data (float a, float b) {
                width = a;
                height = b;  }
};
class Rectangle: public Shape {
        public:
        float area () {
                return (width * height);   }
};
class Triangle: public Shape {
        public:
        float area () {
                return (width * height / 2); }
};
```

One More Example of Hierarchical Inheritance-1

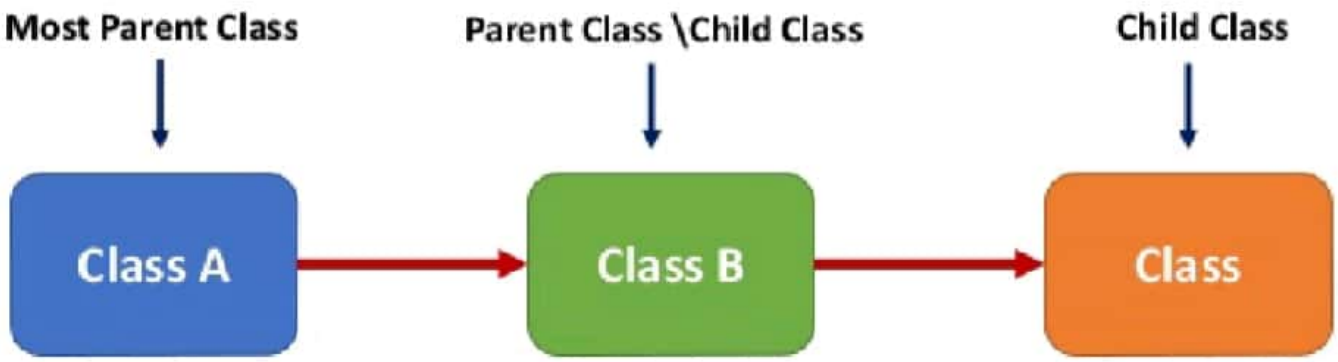
```
int main ()
{
        //Creating an Object of Class Rectangle
        Rectangle rect;
        //Creating an Object of Class Triangle
        Triangle tri;
        rect.set_data (5,3);
        tri.set_data (2,5);
        cout<<"------------Result------------"<<endl<<endl;
        cout <<"Area of Rectangle is: "<<rect.area() << endl;
        cout <<"Area of Triangle is: "<<tri.area() << endl;
return 0;
}
```

Types of Inheritance

- **Multilevel Inheritance**
- In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.

Types of Inheritance

- **Multilevel Inheritance**
- In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.



Multilevel Inheritance

- When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent classes, such inheritance is called **Multilevel Inheritance**. The level of inheritance can be extended to any number of level depending upon the relation. Multilevel inheritance is similar to relation between grandfather, father and child.
- **For Example,**
- Student is derived from person and person is derived from class living things.
- Car is derived from vehicle and vehicle is derived from machine.

Syntax of Multilevel Inheritance

```
class base_classname
{
   properties;
   methods;
};
class intermediate_classname:visibility_mode base_classname
{
   properties;
   methods;
};
class child_classname:visibility_mode intermediate_classname
{
   properties;
   methods;
};
```

Multilevel Inheritance

```
class A
{

};

class B: public A
{


};
class C: public B
{


};
```

Multilevel Inheritance

```
class A
{

};

class B: public A
{

};
class C: public B
{

};
```

Most Parent Class

Parent Class of C and Child Class of A

Child Class of B

Example of Multilevel Inheritance

```
class A {
   public:  void display() {
        cout<<"Base Class Content";
    }
};
class B : public A {

};
class C : public B {

};
int main() {
   C c;
   c.display();
   return 0;
}
```

Example of Multilevel Inheritance

```
class A {
  public:  void display() {
      cout<<"Base Class Content";
  }
};
class B : public A {

};
class C : public B {

};
int main() {
   C c;
   c.display();
   return 0;
}
```

Class B inherit the Member Function from Class A

After inheritance Class B Look Like this

```
class B : public A {
public:  void display() {
      cout<<"Base Class Content";
   }
};
```

Example of Multilevel Inheritance

```
class A {
  public:  void display() {
      cout<<"Base Class Content";
  }
};
class B : public A {

};
class C : public B {

};
int main() {
   C c;
   c.display();
   return 0;
}
```

After inheritance Class C Look Like this

```
class C : public B {
public:  void display() {
      cout<<"Base Class Content";
   }
};
```

Class C inherit the Member Function from Class B

Example of Multilevel Inheritance

```
class A {
   public:  void display() {
        cout<<"Base Class Content";
    }
};
class B : public A {

};
class C : public B {

};
int main() {
   C c;
   c.display();
   return 0;
}
```

Call the display function of class C. This Display function Class C inherit from Class B and Class B inherit from Class A

Base Class Content

One More Example of Multilevel Inheritance-1

```
class grandfather {
 int age;
 char name [20];

 public:
  void get() {
   cout << "Enter your grand father's Name : ";
   cin >> name;
   cout << "Enter your grand father's Age : ";
   cin >> age;
  }
  void show() {
   cout << "\n Your grand father's name is " << name;
   cout << "\n Your grand father's age is " << age;
  }
};
```

One More Example of Multilevel Inheritance-2

```
class father : public grandfather {
  int age;
  char name [20];
  public:
   void get() {
    cout << "Enter your father's Name : ";
    cin >> name;
    cout << "Enter your father's Age : ";
    cin >> age;
   }
   void show() {
    cout << "\n Your father's name is " << name;
    cout << "\n Your father's age is " << age;
   }
};
```

One More Example of Multilevel Inheritance-3

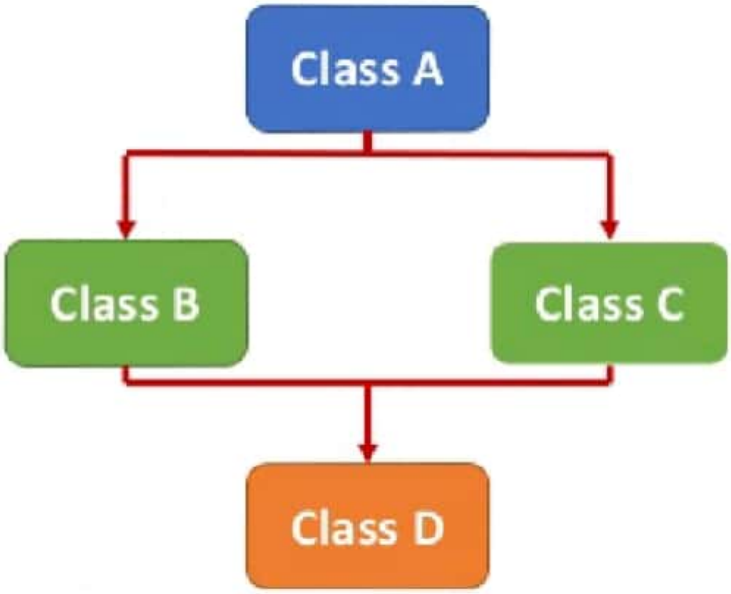
```
class son : public father {
            int age; char name [20];
public: void get() {
    grandfather  :: get();
    father :: get();
    cout << "Enter the child's Name : ";
    cin >> name;
    cout << "Enter the child's Age : ";
    cin >> age;
  }
  void show() {
    grandfather  :: show();
    father :: show();
    cout << "\n Child's name is " << name;
    cout << "\n Child's age is " << age;
  }
};
```

One More Example of Multilevel Inheritance-4

```
int main ()
{
  //Creating an object of Class Son
  son s;
  s.get();
  s.show();
}
```

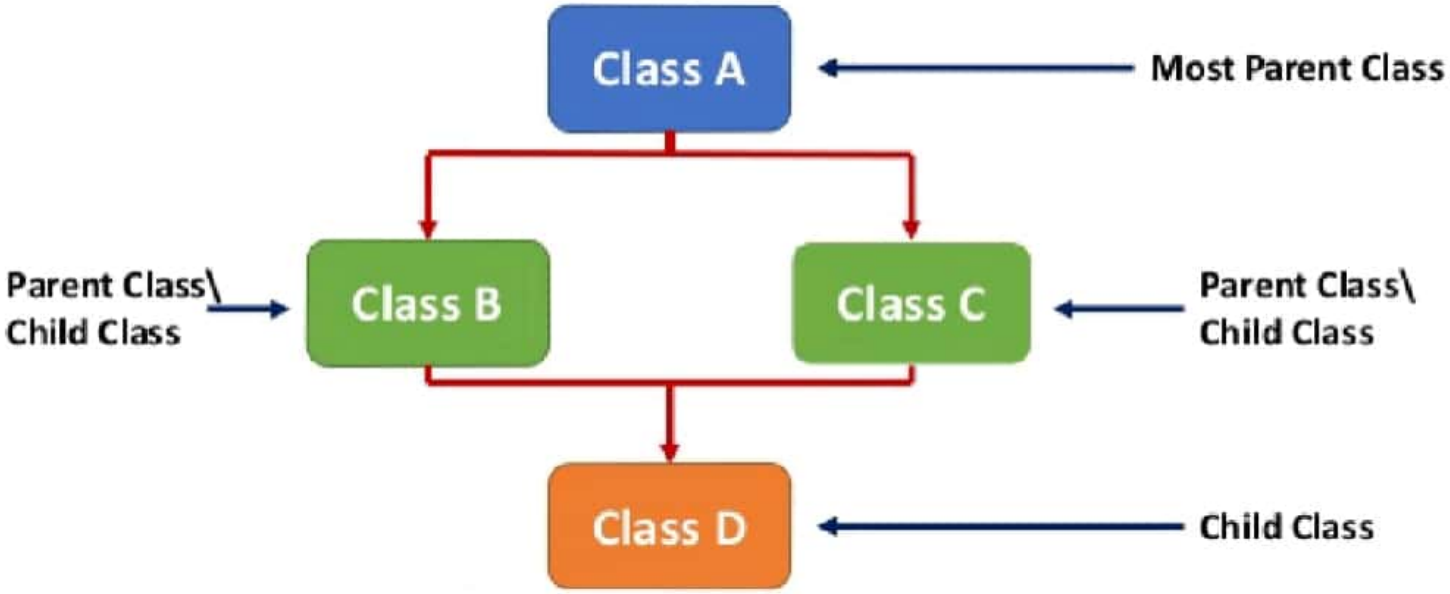
Types of Inheritance

- **Hybrid Inheritance**
- Hybrid Inheritance is a method where one or more types of inheritance are combined together and used.
- Example of Hybrid Inheritance is combination of Hierarchical and Multilevel Inheritance.



Types of Inheritance

- **Hybrid Inheritance**
- Hybrid Inheritance is a method where one or more types of inheritance are combined together and used.
- Example of Hybrid Inheritance is combination of Hierarchical and Multilevel Inheritance.



Hybrid Inheritance Example

```
class A
{

};

class B: public A
{
};
class C: public A
{
};
class D: public B, public C
{
};
```

This is a mixture of Hierarchical and Multilevel inheritance so called hybrid inheritance

Hybrid Inheritance Example

```
class A
{

};

class B: public A
{
};
class C: public A
{
};
class D: public B, public C
{
};
```

Most Parent Class

Class B:: Parent Class of D and Child Class of A

Class C:: Parent Class of D and Child Class of A

Child Class

Example of Hybrid Inheritance-1

```
class student {//base class derivation
 protected:
  int r_no;
 public:
  void getRollno()
  {
   cout << "Enter the roll number of student : ";
   cin >> r_no;
  }
  void putRollno()
  {
   cout << "\nRoll Number -: " << r_no << "\n";
  }
};
```

Example of Hybrid Inheritance-2

```
class test : public student {//intermediate base class
 protected:
  int part1, part2;
 public:
  void getMarks() {
   cout << "Enter the marks of student in SA 1 : ";
   cin >> part1;
   cout << "Enter the marks of student in SA 2 : ";
   cin >> part2;
  }
  void putMarks() {
   cout << "Marks Obtained : " << "\n";
          cout << "  Part 1 -: " << part1;
   cout << "\n  Part 2 -: " << part2 << "\n";
  }
};
```

Example of Hybrid Inheritance-3

```
class sports
{
 protected:
  int score;
 public:
  void getSportsMarks()
  {
   cout << "Enter the marks in Physical Eduction : ";
   cin >> score;
  }
  void putSportsMarks()
  {
   cout << "Additional Marks : " << score << "\n \n";
  }
};
```

Example of Hybrid Inheritance-4

```
//derived from test and sports
class result : public test, public sports
{
  int total;
  public:
   void display ()
   {
     total = part1 + part2 + score;
     putRollno();
     putMarks();
     putSportsMarks();

     cout << "Total Score : " << total ;
   }
};
```

Example of Hybrid Inheritance-5

```
int main ()
{
 //Creating an object of class result
  result s1;
  s1.getRollno();
  s1.getMarks();
  s1.getSportsMarks();
  s1.display();

  return 0;
}
```