<u>Inheritance:</u>- It is a technique by which we can define a new class on the bases of some existing class. The existing class is known as base class that is super class and the new class is known as derived class that is sub class.

The mechanism that allows us to extend the definition of a class without making any physical changes to the existing class is inheritance.

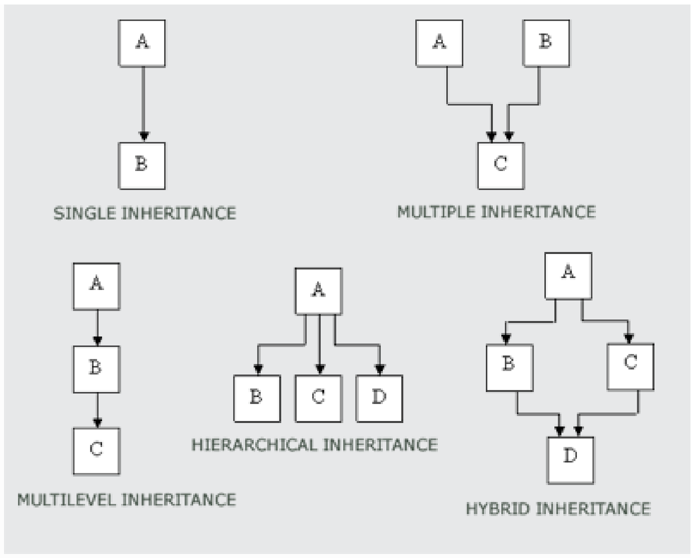
Inheritance lets you create new classes from existing class. Any new class that you create from an existing class is called **derived class**; existing class is called **base class**.

The inheritance relationship enables a derived class to inherit features from its base class. Furthermore, the derived class can add new features of its own. Therefore, rather than create completely new classes from scratch, you can take advantage of inheritance and reduce software complexity.

- Reusability is the basic concept of inheritance.
- In this concept, properties of one class will be extended by another class.
- A class whose properties are inherited is said to be base-class/ parent-class/ super-class.
- A class which inherits/ extends properties of another class is called as derived-class/ child-class/ sub-class.

Syntax:
New class <class name> : visibility mode <existing class>, visibility mode <existing class>
There are five types of Inheritance

1).Single Inheritance: It is the inheritance hierarchy wherein one derived class inherits from one base class.
2).**Multiple Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from multiple base class(es)
3).**Hierarchical Inheritance:** It is the inheritance hierarchy wherein multiple subclasses inherit from one base class.
4).**Multilevel Inheritance:** It is the inheritance hierarchy wherein subclass acts as a base class for other classes.
5).**Hybrid Inheritance:** The inheritance hierarchy that reflects any legal combination of other four types of inheritance.
<u>Advantage of Inheritance</u>:-
1. Reusability
2. Less time required (time saving)
3. Easy to debug


A
B
SINGLE INHERITANCE
A
B
C
MULTIPLE INHERITANCE
A
B
C
MULTILEVEL INHERITANCE
A
B
C
D
HIERARCHICAL INHERITANCE
A
B
C
D
HYBRID INHERITANCE

Single inheritance

```
#include<iostream>
class ABC
{
  char name[20];
  int age;
};
class abc : public ABC
{
  float height;
  float weight;
  public:
  void getdata()
  {
    cout<<”enter name and age “;
    cin>>name>>age;
    cout<<”enter height and weight “;
    cin>>height>>weight;
  }
  void show()
  {
    cout<<”name=”<<name<<endl;
    cout<<”age=”<<age<<endl;
    cout<<”height=”<<height<<endl;
    cout<<”weight=”<<weight<<endl;
  }
};
main()
{
  abc x;
  x.getdata();
  x.show();
}
```

Multiple inheritance

```
#include<iostream>
using namespace std’
class A
{
 int a;
};
class B
{
 int b;
};
class C
{
 int c;
};
class D : public A, B, C
{
 int d;
 void getdata()
 {
  cout<<”enter values of a,b,c,d\n”;
  cin>>a>>b>>c>>d;
 }
 void show()
 {
  cout<<”a=”<<a<<endl;
  cout<<”b=”<<b<<endl;
  cout<<”c=”<<c<<endl;
  cout<<”d=”<<d<<endl;
 }
};
main()
{
 D x;
 x.getdata();
 x.show();
}
```

multi-level inheritance

```
#include<iostream>
using namespace std;
class A
{
  public:
  void display1()
  {
    cout<<”hello”<<endl;
  }
};
class B :public A
{
  public:
  void display2()
  {
    cout<<”world”<<endl;
  }
};

Class C: public B
{
  public:
  void display3()
  {
    cout<<”hai”;
  }
};
main()
{
  C c1;
  C1.diaplay1();
  C1.display2();
}
```

hierarchical inheritance

```
#include<iostream>
using namespace std;
class A
{
  public:
  void display1()
  {
    cout<<”hello”<<endl;
  }
};
class B :public A
{
  public:
  void display2()
  {
    cout<<”world”<<endl;
  }
};
class C: public A
{
  public:
  void display3()
  {
    cout<<”hai”;
  }
};

main()
{
  B b1;
  b1.display1();
  b1.display2();
  C c1;
  c1.display1();
  c1.display3();
}
```

hybrid inheritance

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

class A
{
  public:
  void display1()
  {
    cout<<”hello”<<endl;
  }
};

class B :public A
{
  public:
  void display2()
  {
    cout<<”world”<<endl;
  }
};

class C
{
  public:
  void display3()
  {
    cout<<”hai”;
  }
};

class D:public B, public C
{
  public:
  void display4()
  {
    cout<<”bye”;
  }
};

main()
{
  D d1;
  d1.display1();
  d1.display2();
  d1.display3();
  d1.display4();
}
```

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.

We can summarize the different access types according to who can access them in the following way:

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

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.

Type of Inheritance:
When deriving a class from a base class, the base class may be inherited through public, protected orprivate inheritance. The type of inheritance is specified by the access-specifier as explained above.

We hardly use protected or private inheritance, but public inheritance is commonly used. While using different type of inheritance, following rules are applied:

- Public Inheritance: When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public andprotected members of the base class.

- Protected Inheritance: When deriving from a protected base class, public and protectedmembers of the base class become protected members of the derived class.

- Private Inheritance: When deriving from a private base class, public and protected members of the base class become private members of the derived class.