Introduction

Inheritence is a process of obtaining properties and characteristics(variables and methods) of another class. In this hierarchical order, the class which inherits another class is called subclass or child class, and the other class is the parent class. Inheritance is categorized based on the hierarchy followed and the number of parent classes and subclasses involved.

It is an excellent representation of relationships in the real world.

It allows code reuse. It doesn't require us to create the same code repeatedly and again. It also allows us to add options to an existing class without having to modify the existing code.

It is a transitive nature, meaning that if B is inherited from another class A, all the subclasses belonging to B will inherit directly from class A.

Types of Inheritance

There are five types of inheritances:

Single Inheritance

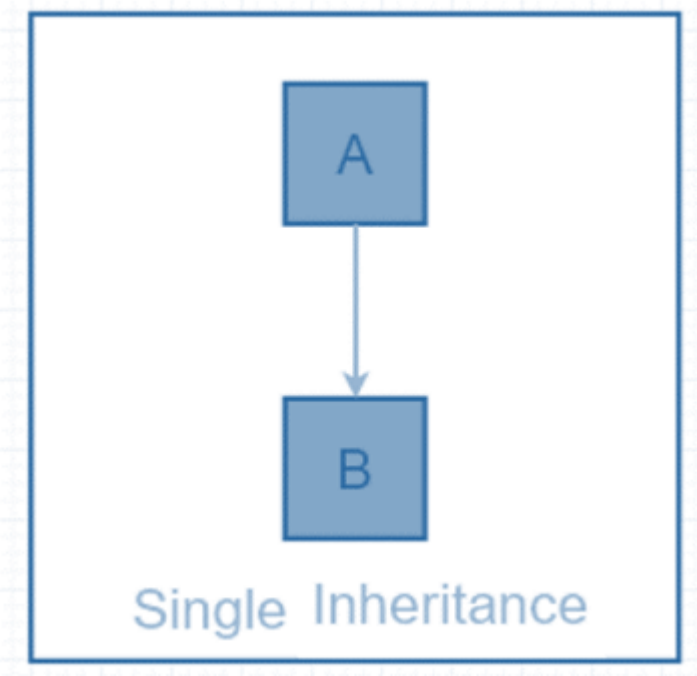
Multiple Inheritance

Multilevel Inheritance

Hierarchical Inheritance

Hybrid Inheritance

Single Inheritance



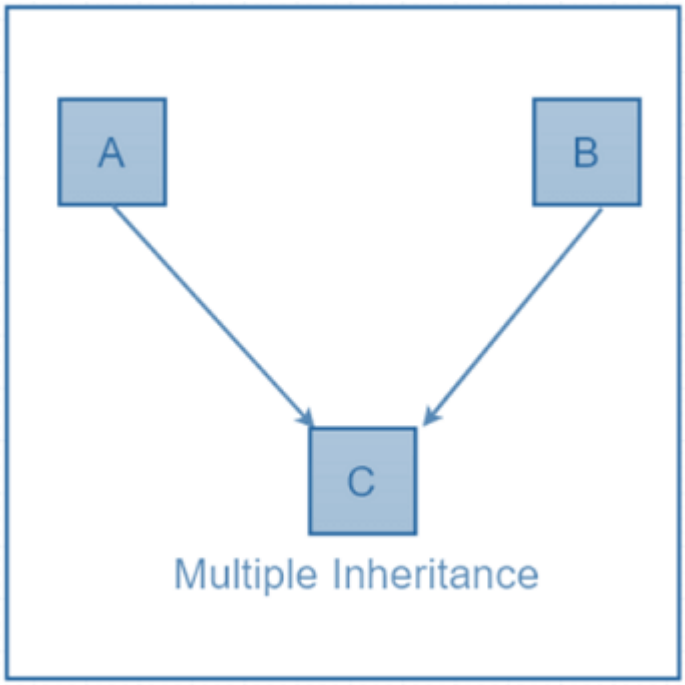
This type of inheritance enables a subclass or derived class to inherit properties and characteristics of the parent class, this avoids duplication of code and improves code reusability.

```
class Above:
    i = 5
    def fun1(self):
        print("Hey there, you are in the parent class")

#subclass
class Below(Above):
    i=10
    def fun2(self):
        print("Hey there, you are in the sub class")

temp1=Below()
temp2=Above()
temp1.fun1()
temp1.fun2()
temp2.fun1()
print(temp1.i)
print(temp2.i)
```

Multiple Inheritance

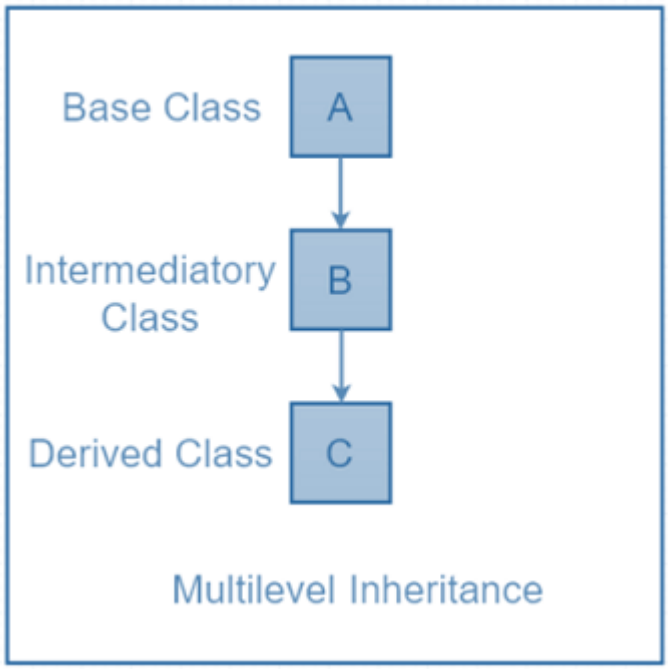


This inheritance enables a child class to inherit from more than one parent class. This type of inheritance is not supported by java classes, but python does support this kind of inheritance. It has a massive advantage if we have a requirement of gathering multiple characteristics from different classes.

```
class A:
    demo1=0
    def fun1(self):
        print(self.demo1)
class B:
    demo2=0
    def fun2(self):
        print(self.demo2)
class C(A, B):
    def fun3(self):
        print("Hey there, you are in the child class")
c = C()
c.demo1 = 10
c.demo2 = 5
c.fun3()
print("first number is : ",c.demo1)
print("second number is : ",c.demo2)
```

Multilevel inheritance

The features that are part of the original class, as well as the class that is derived from it, are passed on to the new class. It is similar to a relationship involving grandparents and children.



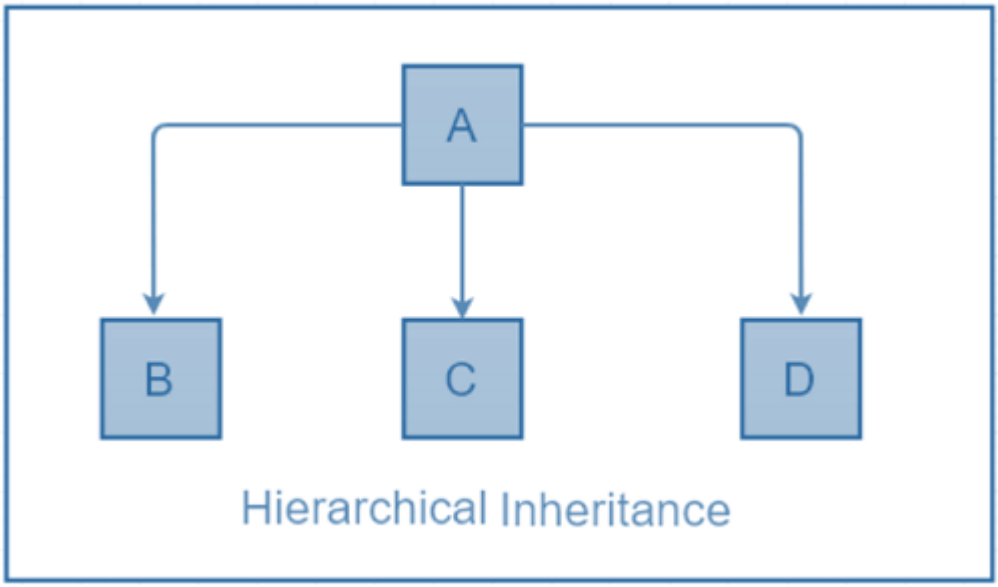

```
class Grandfather1:

    def __init__(self, grandfathername1):
        self.grandfathername1 = grandfathername1
class Father1(Grandfather1):
    def __init__(self, fathername1, grandfathername1):
        self.fathername1 = fathername1
        Grandfather1.__init__(self, grandfathername1)
class Son1(Father1):
    def __init__(self,sonname1, fathername1, grandfathername1):
        self.sonname1 = sonname1
        Father1.__init__(self, fathername1, grandfathername1)

    def print_name(self):
        print('Grandfather name is :', self.grandfathername1)
        print("Father name is :", self.fathername1)
        print("Son name is :", self.sonname1)
s1 = Son1('John', 'John Jr', 'John Jr Jr')
print (s1.grandfathername1)
s1.print_name()
```

Hierarchical Inheritance

If multiple derived classes are created from the same base, this kind of Inheritance is known as hierarchical inheritance. In this instance, we have two base classes as a parent (base) class as well as two children (derived) classes.




```
class Parent1:
    def func_1(self):
        print ("This function is defined inside the parent class.")

# Derived class1
class Child_1(Parent1):
    def func_2(self):
        print ("This function is defined inside the child 1.")

# Derivied class2
class Child_2(Parent1):
    def func_3(self):
        print ("This function is defined inside the child 2.")

# Driver's code
object1 = Child_1()
object2 = Child_2()
object1.func_1()
object1.func_2()
object2.func_1()
object2.func_3()
```

Hybrid inheritance

Hybrid Inheritance is a blend of more than one type of inheritance. The class is derived from the two classes as in the multiple inheritance. However, one of the parent classes is not the base class. It is a derived class. This feature enables the user to utilize the feature of inheritance at its best. This satisfies the requirement of implementing a code that needs multiple inheritances in implementation.