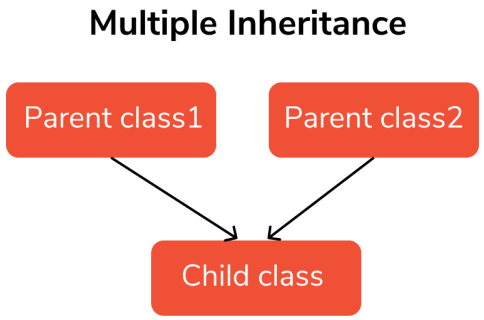
Example:

```
# Multi level inheritance
class Person:
    def __init__(self):
        self.__name=None
    def setName(self,n):
        self.__name=n
    def getName(self):
        return self.__name

class Employee(Person):
    def __init__(self):
        super().__init__()
        self.__job=None
    def setJob(self,j):
        self.__job=j
    def getJob(self):
        return self.__job

class SalariedEmployee(Employee):
    def __init__(self):
        super().__init__()
        self.__salary=None

    def setSalary(self,s):
        self.__salary=s
    def getSalary(self):
        return self.__salary


emp1=SalariedEmployee()
emp1.setName("Naresh")
emp1.setJob("CEO")
emp1.setSalary(500000)
print(f'Name {emp1.getName()}')
print(f'Job {emp1.getJob()}')
print(f'Salary {emp1.getSalary()}')
```

Output:
Name Naresh
Job CEO
Salary 500000

Multiple Inheritance

If a class is derived from more than one base class, it is called multiple inheritance.



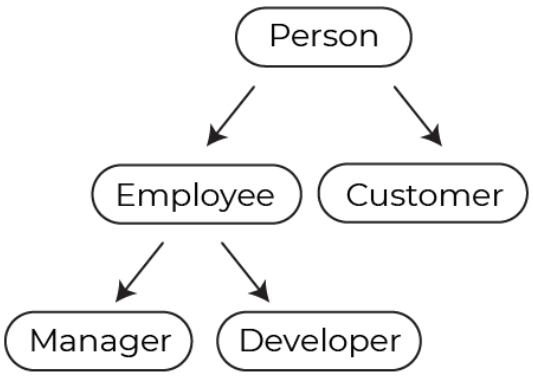
Example:

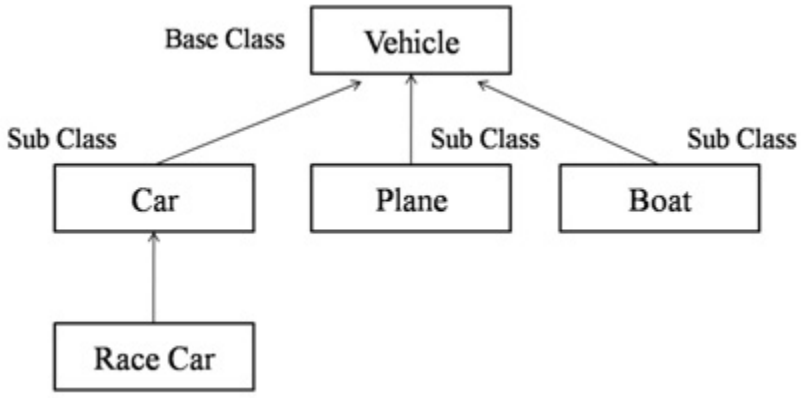
```
class Person:
   def __init__(self):
        self.__name=None
   def setName(self,n):
        self.__name=n
   def getName(self):
        return self.__name

class Faculty:
    def __init__(self):
         self.__subject=None
    def setSubject(self,s):
         self.__subject=s
    def getSubject(self):
        return self.__subject

class Member(Person,Faculty):
   def __init__(self):
```

```
        super().__init__()
        Faculty.__init__(self)
        self.__id=None
    def setId(self,i):
        self.__id=i
    def getId(self):
        return self.__id


member1=Member()
member1.setName("RAM")
member1.setSubject("CS")
member1.setId(101)
print(f'Name {member1.getName()}')
print(f'Subject {member1.getSubject()}')
print(f'Member ID {member1.getId()}')
```

Output:
Name RAM
Subject CS
Member ID 101

Hierarchical Inheritance
If more than one class derived from same base class, it is called hierarchical inheritance.





Example:

```
class Person:
    def __init__(self):
        self.__name=None
    def setName(self,n):
        self.__name=n
    def getName(self):
        return self.__name

class Employee(Person):
    def __init__(self):
        super().__init__()
        self.__job=None
    def setJob(self,j):
        self.__job=j
    def getJob(self):
        return self.__job

class Customer(Person):
    def __init__(self):
        super().__init__()
        self.creditLimit=None
    def setCreditLimit(self,c):
        self.creditLimit=c
    def getCreditLimit(self):
        return self.creditLimit
```

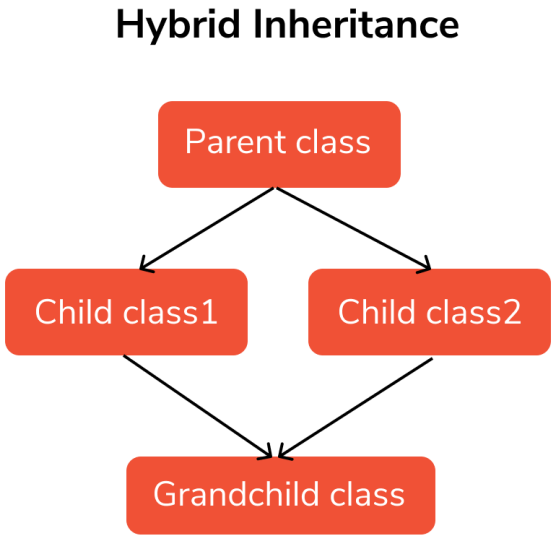
```
emp1=Employee()
cust1=Customer()
emp1.setName("Suresh")
emp1.setJob("Manager")
cust1.setName("Ramesh")
cust1.setCreditLimit(10000)
print(f'Name {emp1.getName()}')
print(f'Job {emp1.getJob()}')
print(f'Name {cust1.getName()}')
print(f'CreditLimit {cust1.getCreditLimit()}')
```

Output:

```
Name Suresh
Job Manager
Name Ramesh
CreditLimit 10000
```

Hybrid Inheritance

If classes are organized using more than one type of inheritance, it is called hybrid inheritance.



Polymorphism
“Poly” means many and “morphism” is forms, defining one thing in many forms is called polymorphism.

In python polymorphism is implemented using,

1. Method Overriding
2. Abstract classes and abstract methods

Advantage of polymorphism is reusability and extensibility.

Method Overriding

Defining instance method in derived class with same name and number of parameters/arguments of method exists in base class is called method overriding.
Method overriding allows providing different implementation of base class method within derived class (OR) it allows modifying or extending functionality of base class method within derived class.