Hierarchical Inheritance

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

Example:

```
class A:   # Parent Class
    def m1(self):
        print("m1 of A class")


class B(A): # Child Class
    def m2(self):
        print("m2 of B class")

class C(A): # Child class
    def m3(self):
        print("m3 of C class")

objc=C()
objc.m1()
objc.m3()

objb=B()
objb.m1()
objb.m2()
```

Output

```
m1 of A class
m3 of C class
m1 of A class
m2 of B class
```

Example:

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

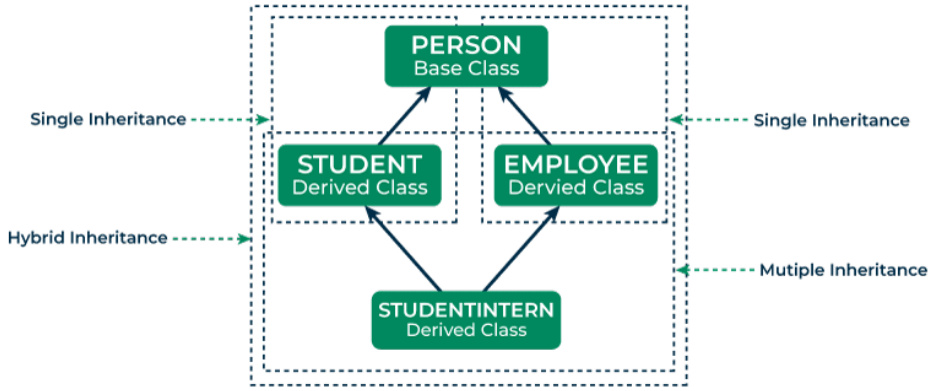
```
        self.__name=n
    def getName(self):
        return self.__name

class Student(Person):
    def __init__(self):
        super().__init__()
        self.__course=None
    def setCourse(self,c):
        self.__course=c
    def getCourse(self):
        return self.__course

class Employee(Person):
    def __init__(self):
        super().__init__()
        self.__salary=None
    def setSalary(self,s):
        self.__salary=s
    def getSalary(self):
        return self.__salary


stud1=Student()
emp1=Employee()

stud1.setName("naresh")
stud1.setCourse("java")

emp1.setName("ramesh")
emp1.setSalary(45000)

print(stud1.getName(),stud1.getCourse())
print(emp1.getName(),emp1.getSalary())
```

Output

```
naresh java
ramesh 45000
```

Hybrid Inheritance

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

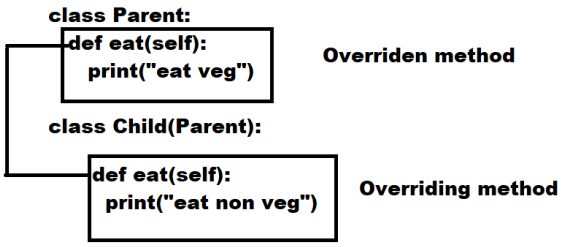


Method Overriding

Defining instance method inside derived class with same name, number of parameters, types of parameters of method defined inside base class is called method overriding.

Use of method overriding

In application development method overriding is done, when sub class wants to modify or extend functionality of base class method (OR) if sub class wants to provide different implementation of method exists in base class.



Example:

```
class A:
    def m1(self):
        print("m1 of A")
    def m2(self):
```

```
        print("m2 of A")


class B(A):
    def m3(self):
        print("m3 of B")
    def m1(self): # m1 method of A class is override
         print("overriding method")


objb=B()
objb.m1()
objb.m2()
objb.m3()
```

Output

```
overriding method
m2 of A
m3 of B
```

Example:

```
class Employee:
    def __init__(self):
        self.__ename=None
        self.__job=None
    def read(self):
        self.__ename=input("EmployeeName :")
        self.__job=input("Job: ")
    def print_info(self):
        print(f'EmployeeName {self.__ename}')
        print(f'EmployeeJob {self.__job}')

class SalariedEmployee(Employee):
    def __init__(self):
        super().__init__()
        self.__salary=None
    def read(self): # Overriding method
        super().read()
        self.__salary=float(input("Salary :"))
```

```
    def print_info(self): # Overriding method
        super().print_info()
        print(f'Employee Salary {self.__salary}')


emp1=SalariedEmployee()
emp1.read()
emp1.print_info()
```

Output

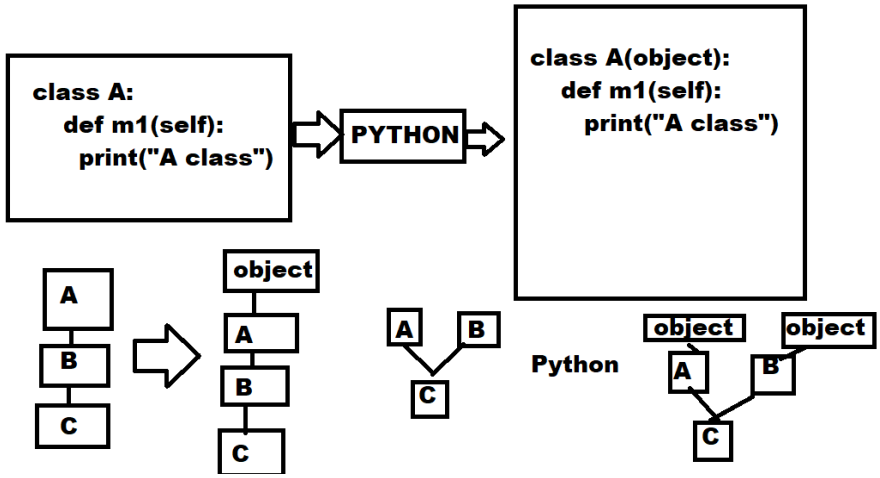
```
EmployeeName :naresh
Job: hr
Salary :45000
EmployeeName naresh
EmployeeJob hr
Employee Salary 45000.0
```

object class

Every class in python inherits automatically one class called object class. (OR) object class is a base class for all classes in python (user defined class or predefined classes)

Object class is having the properties and behavior common for all the classes (OR) these methods are used to manage objects

Methods of object class are magic methods.
Any method in python prefix and suffix with double underscore __ is called magic method. This method is executed automatically when specific operations performed on object.



What is __str__ method?
The __str__() method returns a human-readable, or informal, string representation of an object. This method is called by the built-in print() , str() , and format() functions.

It is a method of object class.

Example:

```
class Product:
    def __init__(self,n,p):
        self.__name=n
        self.__price=p
    def __str__(self):
        return f'{self.__name},{self.__price}'


p1=Product("mouse",300)
c1=complex(1.5,1.2)

print(c1)
print(p1)

print(str(c1))
print(str(p1)) # str(p1.__str__())
```

Output

(1.5+1.2j)
mouse,300
(1.5+1.2j)
mouse,300

What is __repr__ method?
This repr method is executed automatically when repr() function is invoked on object. __repr__ method returns string representation of object. It returns information about how object this class is constructed.
This method is called if __str__ method not override inside subclass.

```
class Employee:
    def __init__(self,en,s):
        self.__ename=en
        self.__salary=s
    def __str__(self): # Overriding method
        return f'{self.__ename},{self.__salary}'
    def __repr__(self): # Overriding method
        return f'Employee({self.__ename},{self.__salary})'



emp1=Employee("naresh",45000)
print(emp1)
print(repr(emp1))
```

Output
naresh,45000
Employee(naresh,45000)