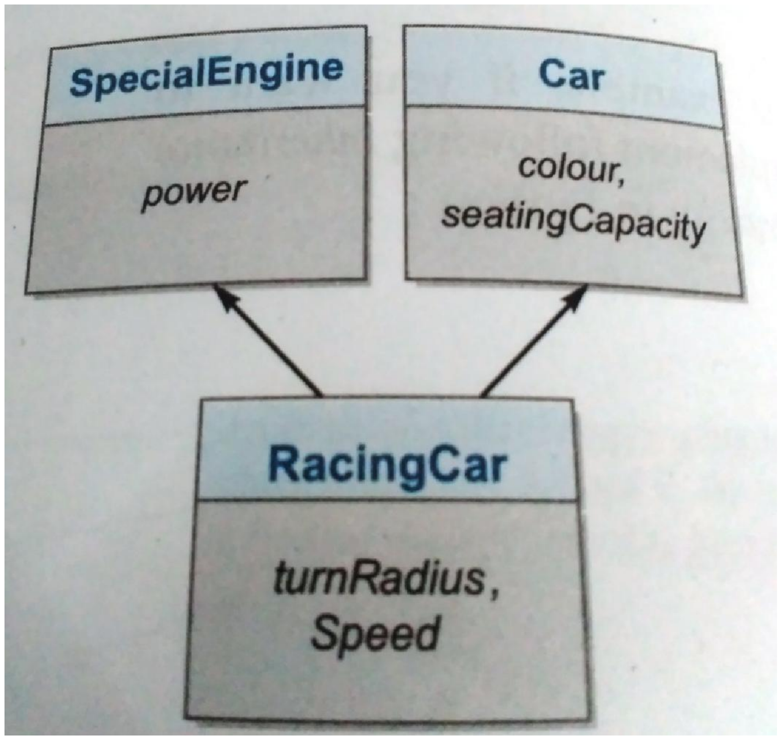
Multiple Inheritance

When a subclass inherit from multiple base classes, it is called multiple Inheritance.

Syntax

```
class<SubClass_Name>(<Parent1_Class_Name>, <Parent2_Class_Name>,…):
        [<class docstring>]
        <suite>
```

Like single inheritance, the subclass in multiple inheritance will inherit the public members from all its base classes. Same example of RacingCar inheriting from Car and SpecialEngine classes has been implemented.



The above mention classes will be implemented in the program below:

```
class SpecialEngine(object):
       def __init__(self, p):
              self.power=p
       def ignition(self):
              print("Engine started!!!")
class Car(object):
       def __init__(self, clr, seats):
              self.color=clr
              self.seats=seats
       def changeGears(self, gr):
              print("Changed to gear ", gr)

       def turn(self, direction):
              print("Turned to ", direction, " direction")
class RacingCar(SpecialEngine, Car):
       def __init__(self, clr, seats, p, tr, spd):
              SpecialEngine.__init__(self, p)
              Car.__init__(self, clr, seats)
              self.turnRadius=tr
              self.speed=spd
              print("Racing car instance created")
       def start(self):
              self.ignition()
              self.changeGears(2)
              print("Racing Car starts- ready to vroom")
#=========================main===============================
rcar = RacingCar('Blue', 2, 750, 6, 200)
rcar.start()
rcar.turn("left")
```

Output:

```
=======================RESTART: E:/multipleInherit.py =======================
Racing car instance created
Engine started!!!
Changed to gear  2
Racing Car starts- ready to vroom
Turned to  left  direction
```

This is proprietary content of SSi Indore, Unauthorized use of this is restricted

Method Overriding

The sub class may modify a method of super class by defining it again. Within the subclass, the original definition is replaced with the new one. This redefinition of a method in the sub class is called overriding the method.

But before overriding a method you should be aware with certain things:

i) You cannot override a private method of base class, ie a method that has a name beginning with two underscores.

ii) The overridden method in the child class preferably should have the same name, signature, and parameters as the one in its parent class.

One reason for overriding parent's methods is because you may want special or different functionality in your subclass.

```
class Parent:  # define parent class
    def myMethod(self):
        print('Calling parent method')

class Child(Parent): # define child class
    def myMethod(self):
        print 'Calling child method'

c = Child()  # instance of child

c.myMethod()  # child calls overridden method
```

When You run the above code the output will be

```
Calling child method
```

One more Example:

```
class A:
    def __init__(self):
        self.val1=10
    def method1(self):
        print("Method1 of A")
class B(A):
    def __init__(self):
        myvar = 20
    def method1(self):
        print("Method of B")
```

This is proprietary content of SSi Indore, Unauthorized use of this is restricted

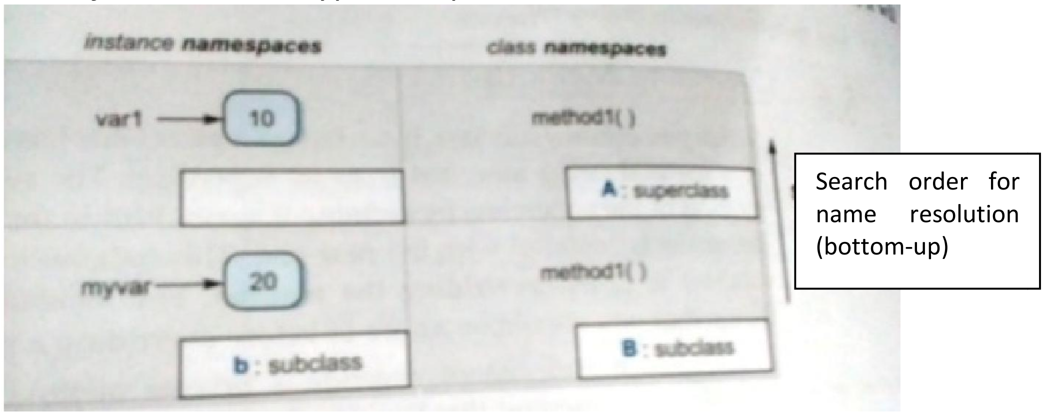
```
    def method2(self):
        self.method1()
#================= main =======================
b = B()
b.method1()
b.method2()
```

Output:

```
>>>
 RESTART:
C:/Users/ssi2/AppData/Local/Programs/Python/Python36-
32/inherit.py
Method of B
Method of B
>>>
```

Python has invoked the method1() defined in class B and not the one defined in class A even though that method has been inherited and available to class B's object.

Let us understand why and how this happened? If we make the hierarchical namespace for object b of class type B as per above code it will look like as shown below:



As you know that when an identifier name(ie, name of an attribute or method) is used or accessed via an object, Python searches in the hierarchical namespace created for the object, in **bottom to top order.**

So, when Python encounter the statement:

```
    b.method1()
```

it will first search for a name method1() in the bottom-most partition of the hierarchical namespace - and it finds a definition of method1() in the class namespace of subclass B and execute it, hence the above output.

This is proprietary content of SSi Indore, Unauthorized use of this is restricted