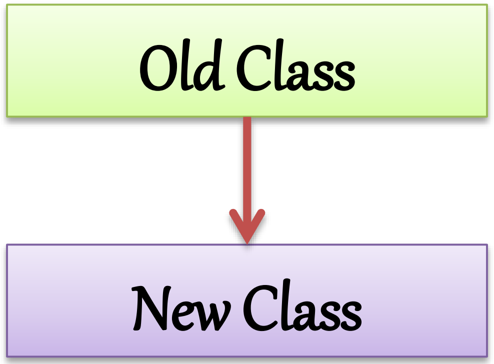
Inheritance

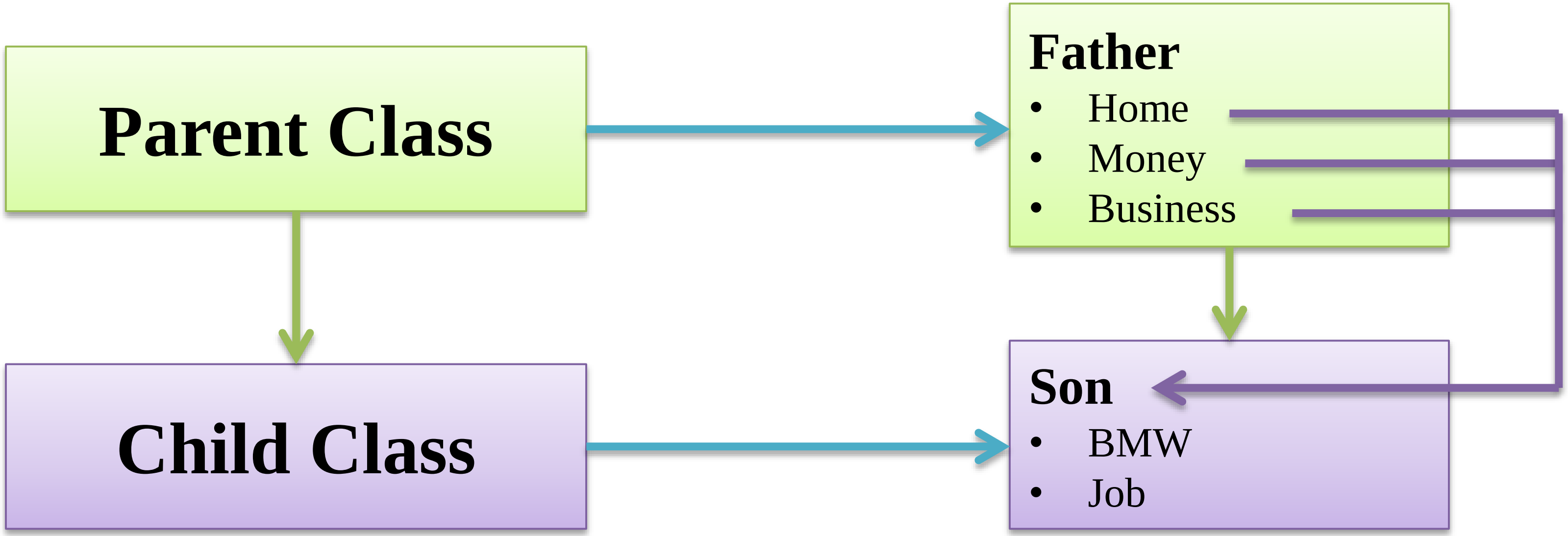
The mechanism of deriving a new class from an old one (existing class) such that the new class inherit all the members (variables and methods) of old class is called inheritance or derivation.



Super Class and Sub Class

The old class is referred to as the Super class and the new one is called the Sub class.

• Parent Class	- Base Class or Super Class
• Child Class	- Derived Class or Sub Class



Inheritance

- All classes in python are built from a single super class called ‘object’ so whenever we create a class in python, object will become super class for them internally.

```
        class Mobile(object):
        class Mobile:
```

- The main advantage of inheritance is code reusability.

Why do We need inheritance

```
class Employee :
    id = 1
    @classmethod
    def getid(cls):
        return cls.id
    def setname(self, name):
        self.name = name
    def getname(self):
        return self.name
    def setsalary(self, salary):
        self.salary = salary
    def getsalary(self):
        return self.salary
    def setovertime(self, ot):
        self.ot = ot
    def getovertime(self):
        return self.ot
```

```
class Manager :
    id = 1
    @classmethod
    def getid(cls):
        return cls.id
    def setname(self, name):
        self.name = name
    def getname(self):
        return self.name
    def setsalary(self, salary):
        self.salary = salary
    def getsalary(self):
        return self.salary
    def setseniorname(self, sname):
        self.sname = sname
    def getseniorname(self):
        return self.sname
```

Parent Class

```
class Employee :
      id = 1
      @classmethod
      def getid(cls):
           return cls.id
     def setname(self, name):
          self.name = name
     def getname(self):
           return self.name
    def setsalary(self, salary):
           self.salary = salary
     def getsalary(self):
           return self.salary
    def setovertime(self, ot):
           self.ot = ot
     def getovertime(self):
           return self.ot
```

Child Class

```
class Manager :
     def setsalary(self, salary):
           self.salary = salary
     def getsalary(self):
           return self.salary
     def getseniorname(self, sname):
           self.sname = sname
     def getseniorname(self):
           return self.sname
```

Type of Inheritance

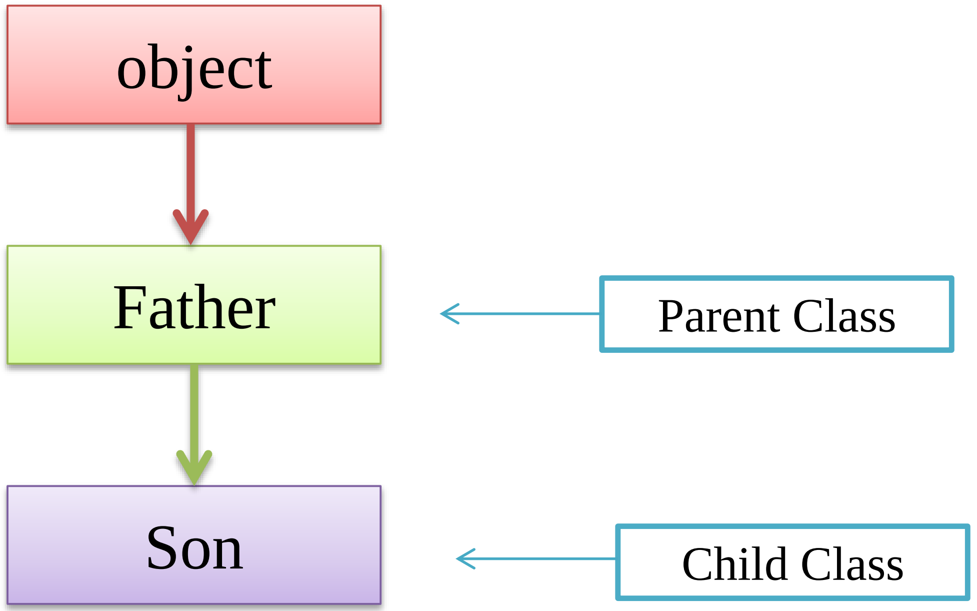
- Single Inheritance
- Multi-level Inheritance
- Hierarchical Inheritance
- Multiple Inheritance

Declaration of Child Class

```
class ChildClassName (ParentClassName) :
        members of Child class


class Mobile (object) :
        members of Child class


class Mobile :
        members of Child class
```

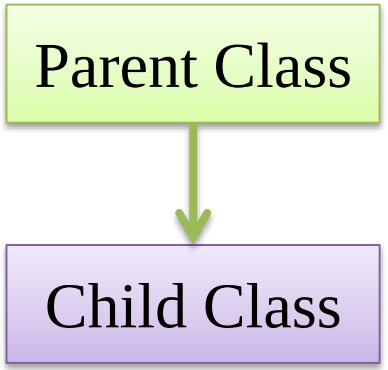
Single Inheritance

If a class is derived from one base class (Parent Class), it is called Single Inheritance.



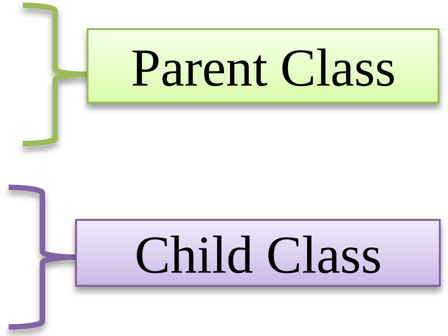
Syntax:-

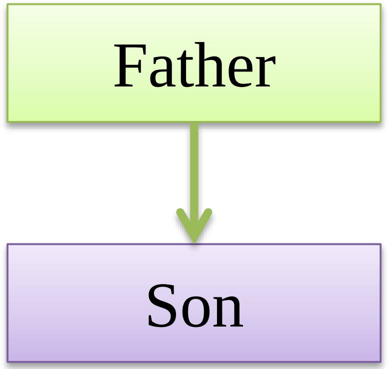
```
class ParentClassName(object):
        members of Parent Class

class ChildClassName(ParentClassName):
        members of Child Class
```



Example:-

```
class Father:
        members of class Father

class Son (Father):
       members of class Son
```





Inheritance

- We can access Parent Class Variables and Methods using Child Class Object

- We can also access Parent Class Variables and Methods using Parent Class Object

- We can not access Child Class Variables and Methods using Parent Class Object

Constructor in Inheritance

By default, The constructor in the parent class is available to the child class.

```
class Father:
        def __init__(self):
                self.money = 2000
                print("Father Class Constructor")

class Son (Father):
        def disp(self):
                print("Son Class Instance Method:",self.money)

s = Son( )
s.disp()
```

What will happen if we define constructor in both classes ?

Constructor Overriding

If we write constructor in the both classes, parent class and child class then the parent class constructor is not available to the child class.

In this case only child class constructor is accessible which means child class constructor is replacing parent class constructor.

Constructor overriding is used when programmer want to modify the existing behavior of a constructor.

Constructor Overriding

```
class Father:
        def __init__(self):
                self.money = 2000
                print("Father Class Constructor")

class Son(Father):
        def __init__(self):
                self.money = 5000
                print("Son Class Constructor")
        def disp(self):
                print(self.money)

s = Son()
s.disp()
```

How can we call parent class constructor ?

Constructor with super( ) Method

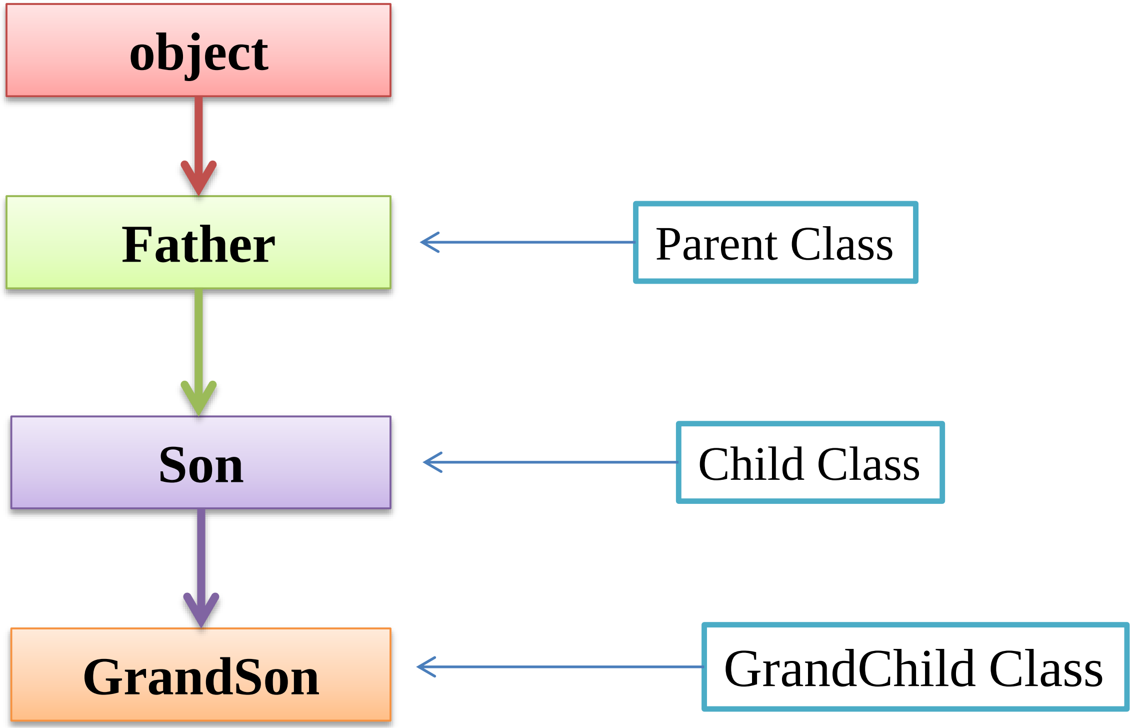
If we write constructor in the both classes, parent class and child class then the parent class constructor is not available to the child class.

In this case only child class constructor is accessible which means child class constructor is replacing parent class constructor.

super ( ) method is used to call parent class constructor or methods from the child class.

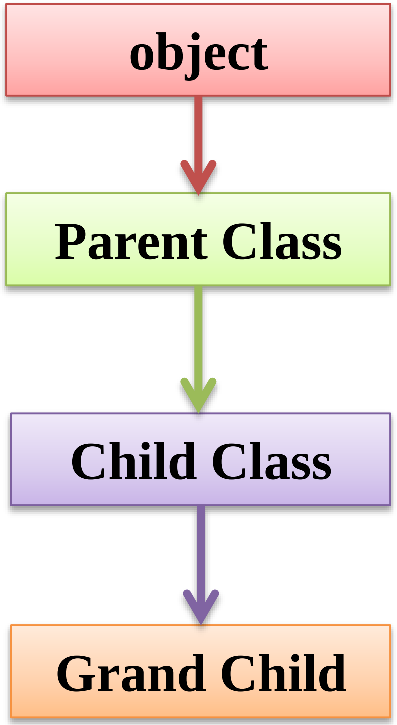
Multi-level Inheritance

In multi-level inheritance, the class inherits the feature of another derived class (Child Class).

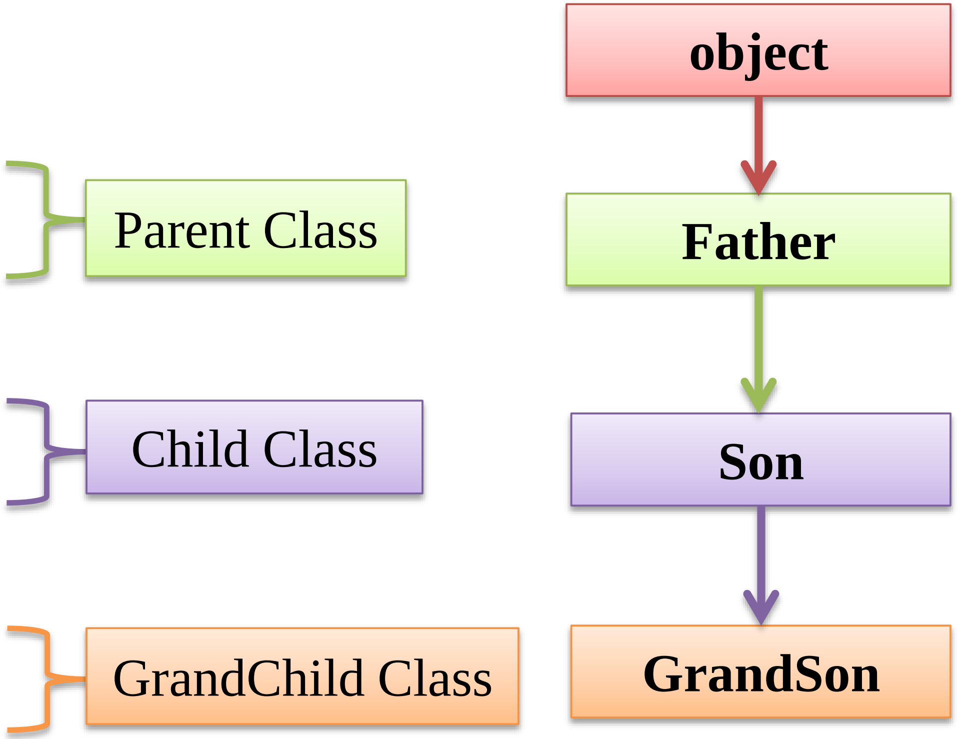


Syntax:-

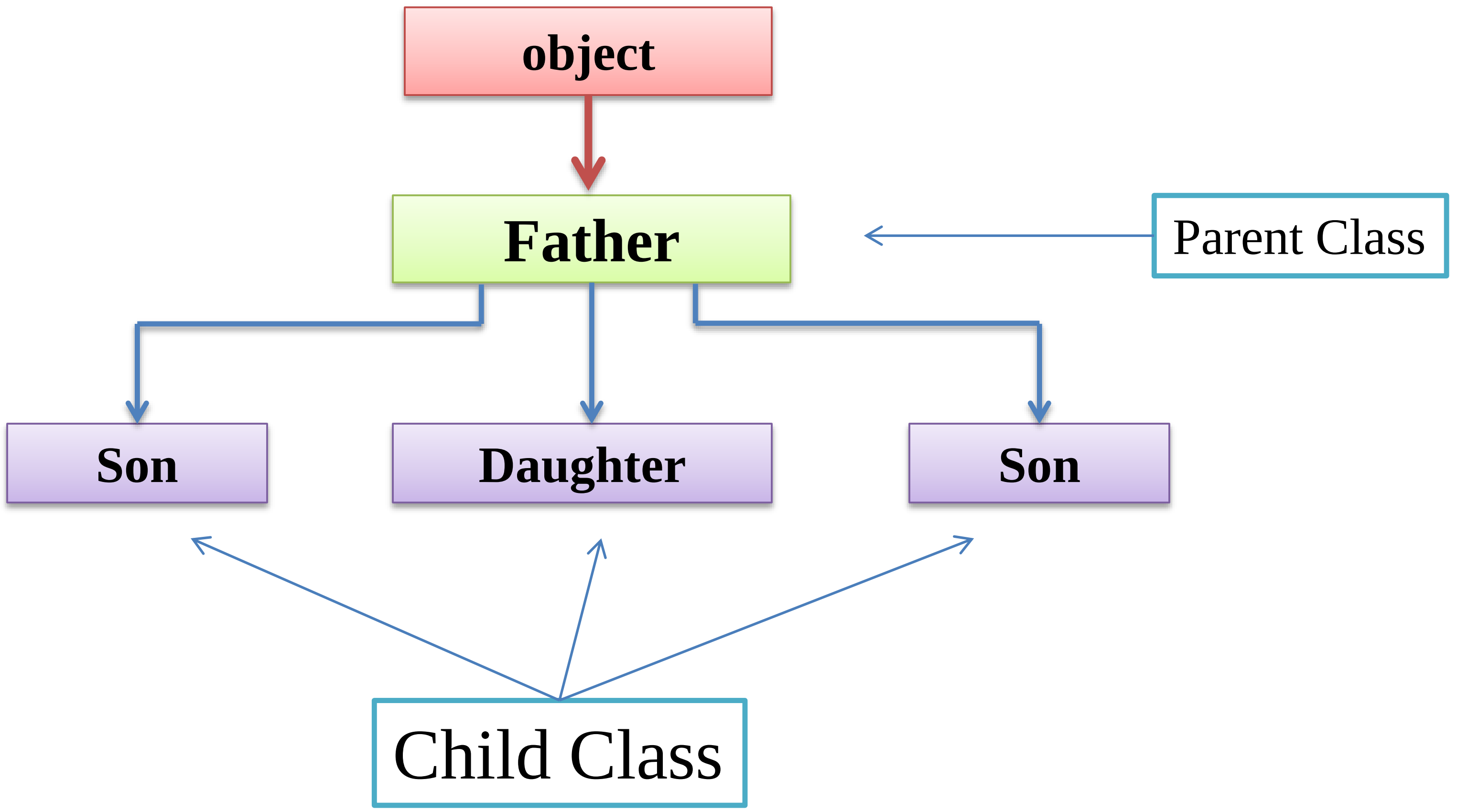
```
class ParentClassName(object):
        members of Parent Class

class ChildClassName(ParentClassName):
        members of Child Class

class GrandChildClassName(ChildClassName):
        members of Grand Child Class
```



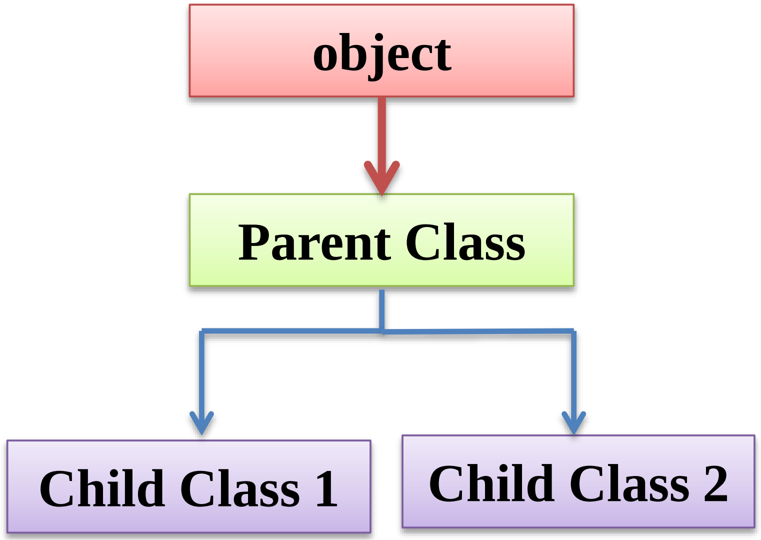
```
class Father (object):
        members of class Father

class Son (Father):
        members of class Son

class GrandSon (Son):
        members of class GrandSon
```



Hierarchical Inheritance



```
Syntax:-
class ParentClassName(object):
        members of Parent Class

class ChildClassName1(ParentClassName):
        members of Child Class 2

class ChildClassName2(ParentClassName):
        members of Child Class 2
```



```
class Father (object):
    members of class Father
```

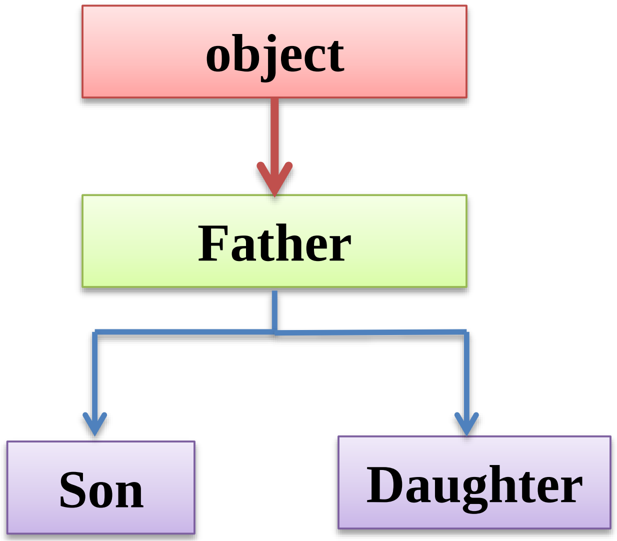
Parent Class

```
class Son (Father):
    members of class Son
```

Child Class

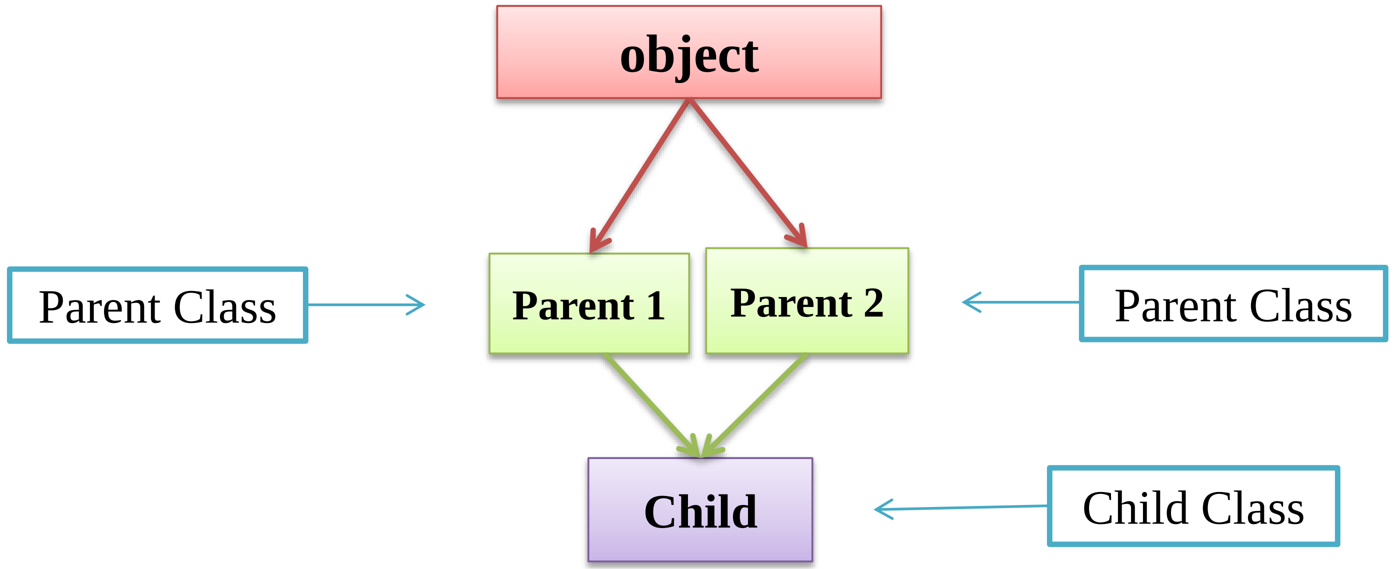
```
class Daughter (Father):
    members of class Daughter
```

Child Class



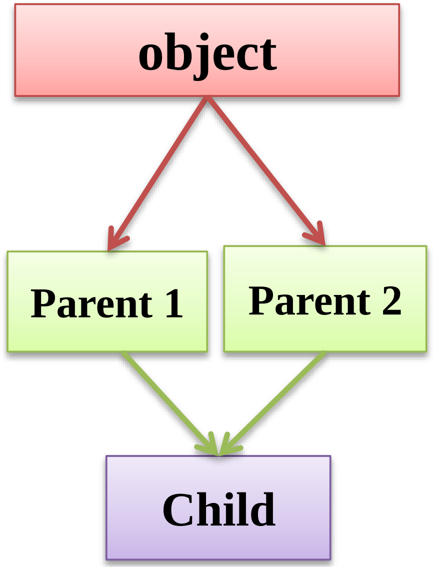
Multiple Inheritance

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



Syntax:-

```
class ParentClassName1(object):
        members of Parent Class

class ParentClassName2(object):
        members of Parent Class

class ChildClassName(ParentClassName1, ParentClassName2):
        members of Child Class
```



```
class Father (object):
    members of class Father
```

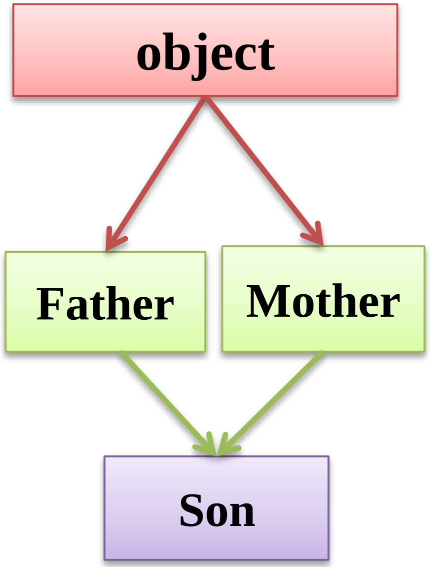
Parent Class

```
class Mother (object):
    members of class Mother
```

Parent Class

```
class Son (Father, Mother):
    members of class Son
```

Child Class



Method Resolution Order (MRO)

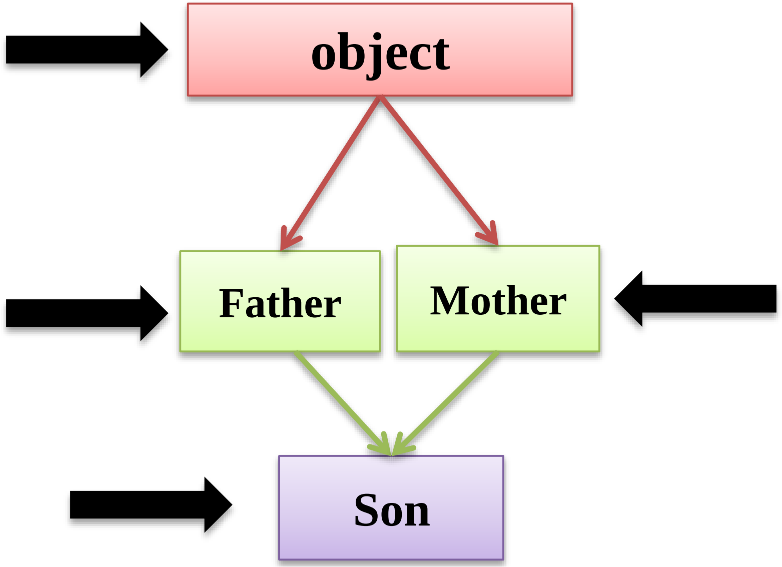
In the multiple inheritance scenario members of class are searched first in the current class. If not found, the search continues into parent classes in depth-first, left to right manner without searching the same class twice.

- Search for the child class before going to its parent class.
- When a class is inherited from several classes, it searches in the order from left to right in the parent classes.
- It will not visit any class more than once which means a class in the inheritance hierarchy is traversed only once exactly.

Method Resolution Order (MRO)

s = Son()

- The search will start from Son. As the object of Son is created, the constructor of Son is called.
- Son has super().__init__() inside his constructor so its parent class, the one in the left side 'Father' class's constructor is called.
- Father class also has super().__init__() inside his constructor so its parent 'object' class's constructor is called.
- Object does not have any constructor so the search will continue down to right hand side class (Mother) of object class so Mother class's constructor is called.
- As Mother class also has super().__inti__() so its parent class 'object' constructor is called but as object class already visited, the search will stop here.


object
Father
Mother
Son