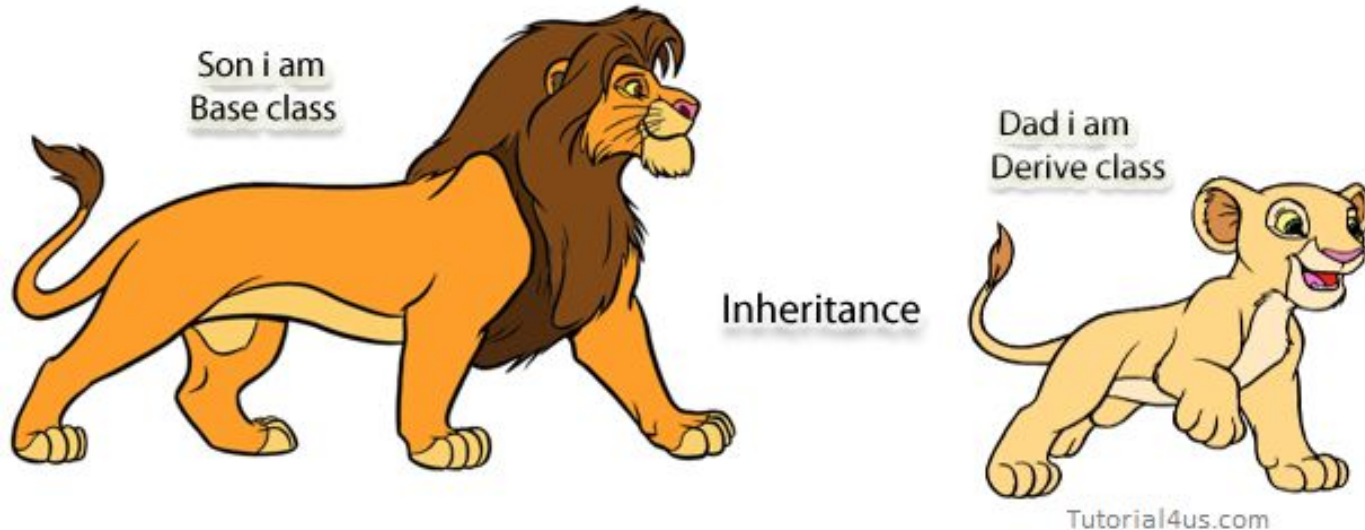
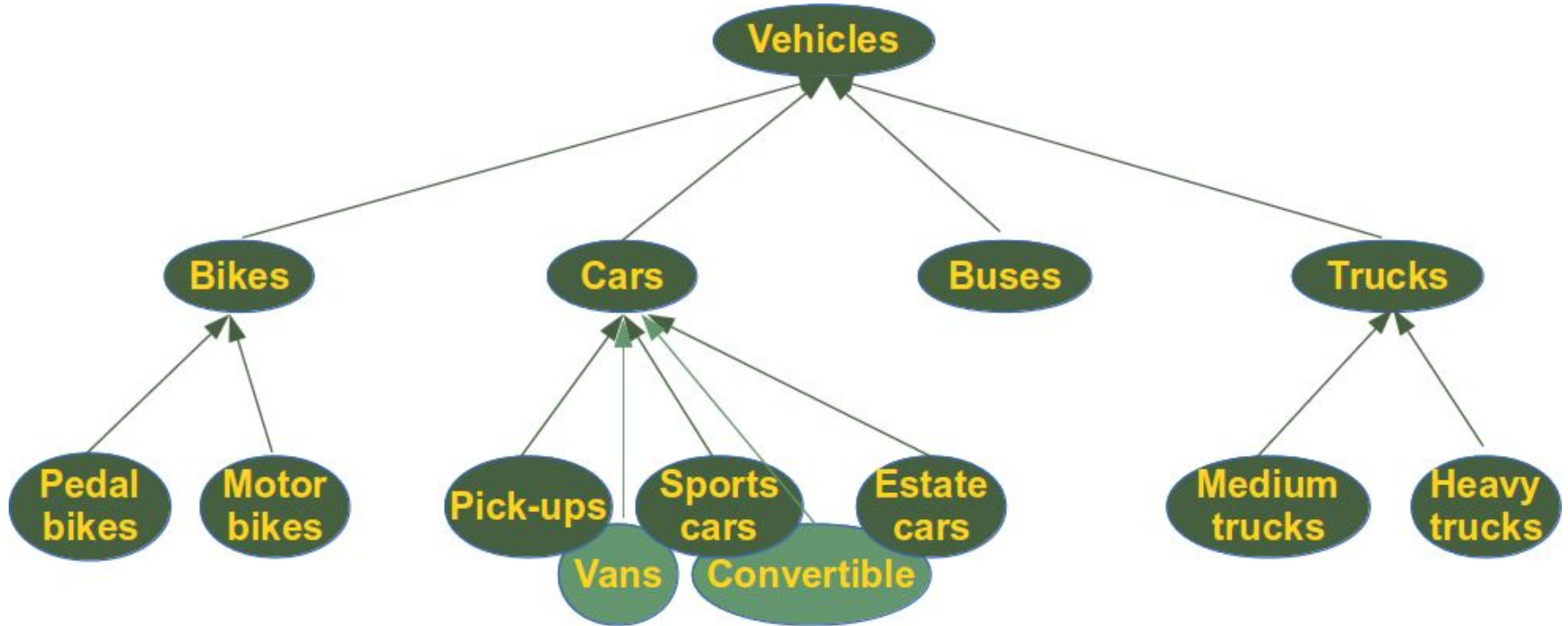


# Inheritance

Md. Nahiyah Uddin



# Inheritance

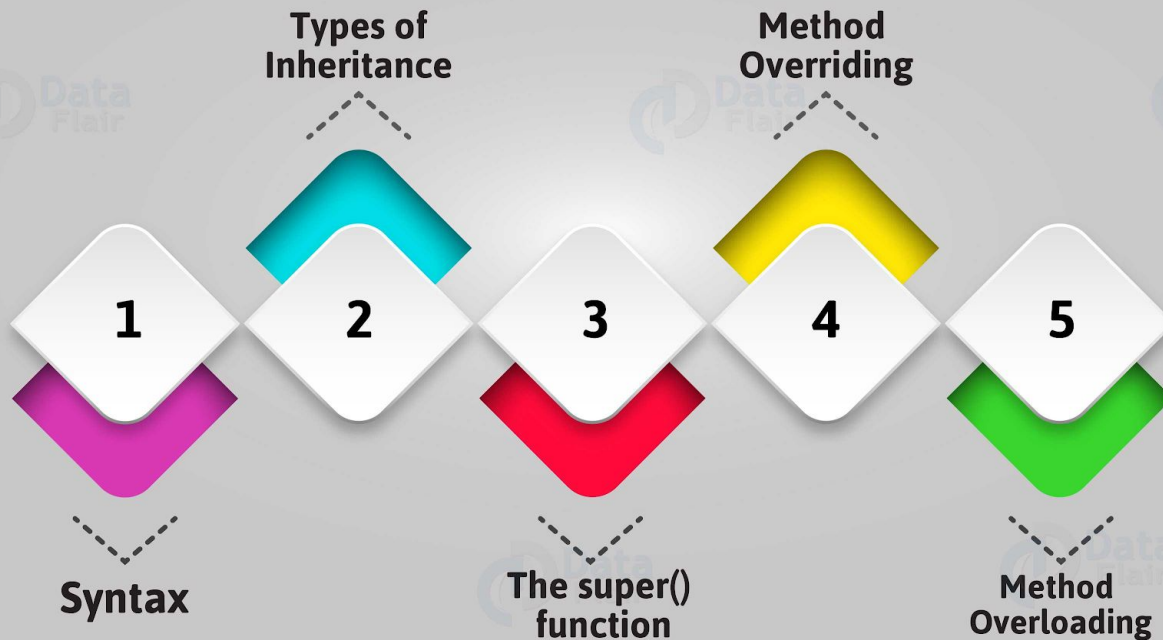


# Definition

**Inheritance is a mechanism in which one class acquires the property of another class.**

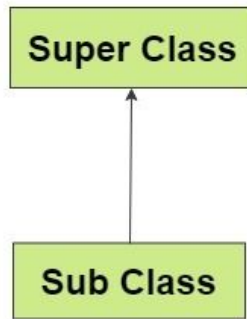
**For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class.**

# Inheritance

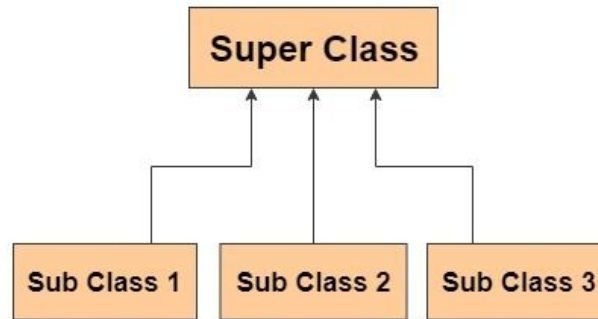


# Types Of Inheritance

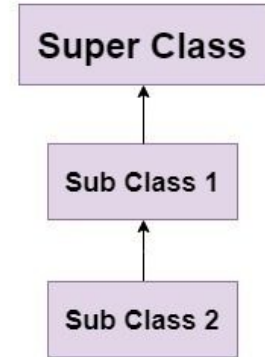
## Single Inheritance



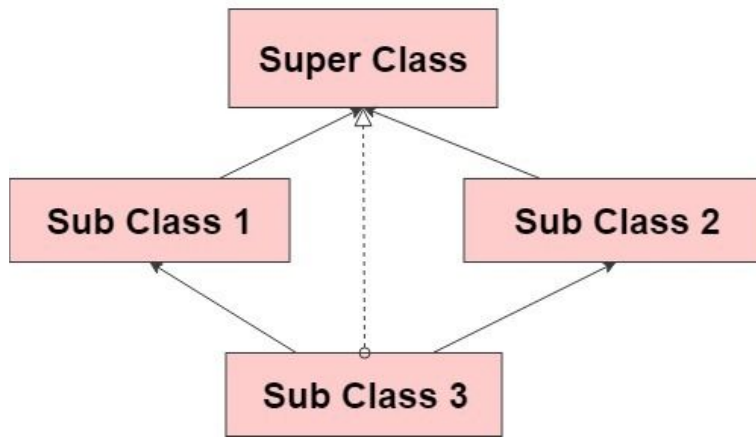
## Hierarchical Inheritance



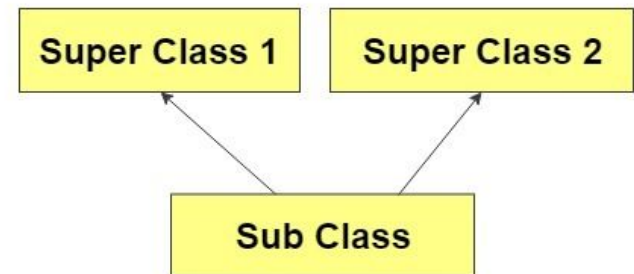
## MultiLevel Inheritance



## Hybrid Inheritance

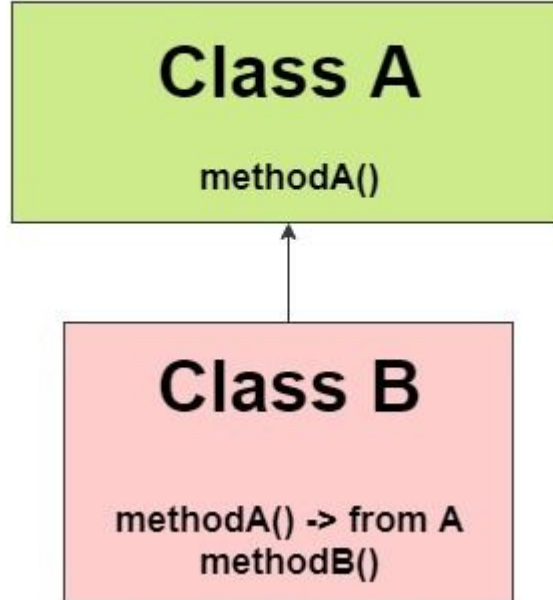


## Multiple Inheritance



# Single Inheritance Structure

## Single Inheritance



# Create a Parent Class

Any class can be a parent class, so the syntax is the same as creating any other class:

## Example

Create a class named `Person`, with `firstname` and `lastname` properties, and a `printname` method:

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)
```

#Use the `Person` class to create an object, and then execute the `printname` method:

```
x = Person("John", "Doe")
x.printname()
```

# Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

## Example

Create a class named `Student`, which will inherit the properties and methods from the `Person` class:

```
class Student(Person):  
    pass
```

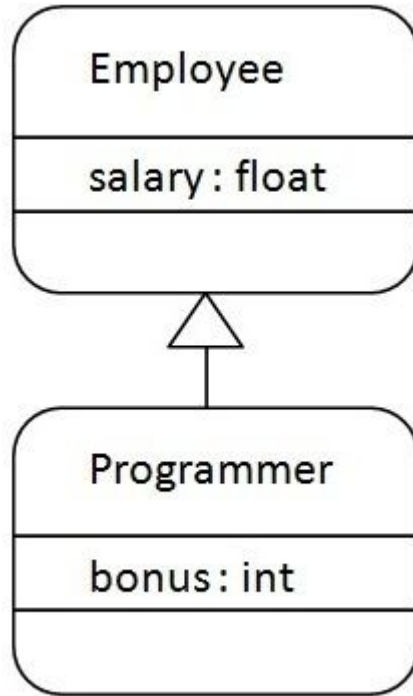
## Example

Use the `Student` class to create an object, and then execute the `printname` method:

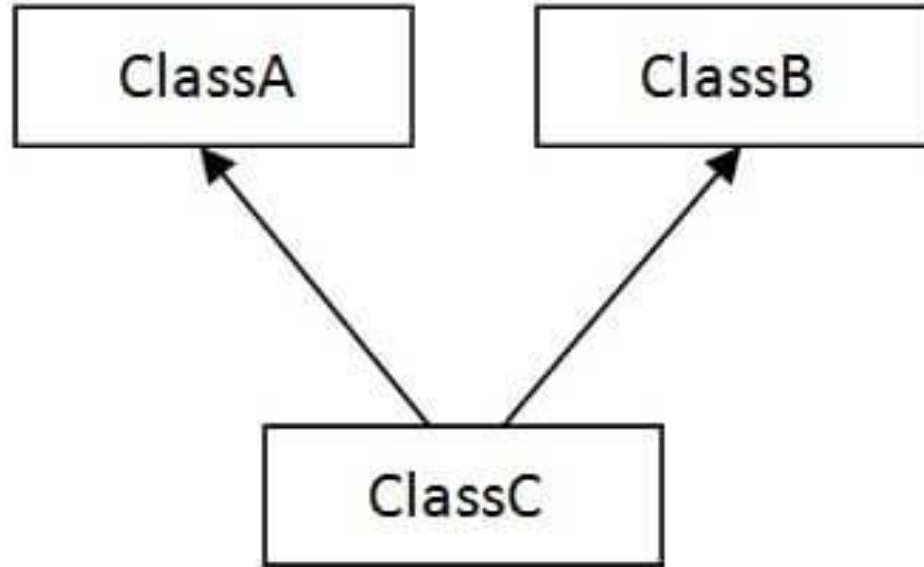
```
x = Student("Mike", "Olsen")  
x.printname()
```



# Single Inheritance Example



# Multiple inheritance Structure



# Multiple inheritance Example

```
# inheritance
class Base1(object):
    def __init__(self):
        self.str1 = "Geek1"
        print "Base1"

class Base2(object):
    def __init__(self):
        self.str2 = "Geek2"
        print "Base2"
```

```
class Derived(Base1, Base2):
    def __init__(self):

        # Calling constructors of Base1
        # and Base2 classes
        Base1.__init__(self)
        Base2.__init__(self)
        print "Derived"

    def printStrs(self):
        print(self.str1, self.str2)

ob = Derived()
ob.printStrs()
```

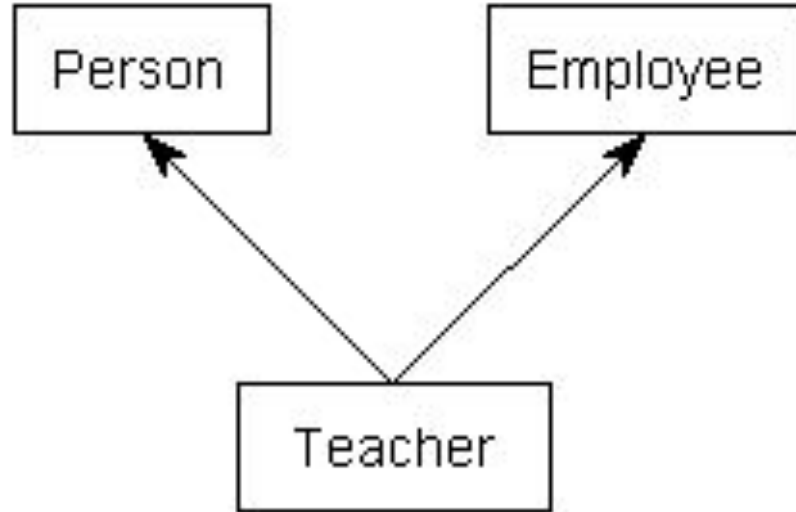
```
# first parent class
class Person(object):
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

# second parent class
class Employee(object):
    def __init__(self, salary, post):
        self.salary = salary
        self.post = post

# inheritance from both the parent classes
class Leader(Person, Employee):
    def __init__(self, name, idnumber, salary, post, points):
        self.points = points
        Person.__init__(self, name, idnumber)
        Employee.__init__(self, salary, post)
        print(self.salary)

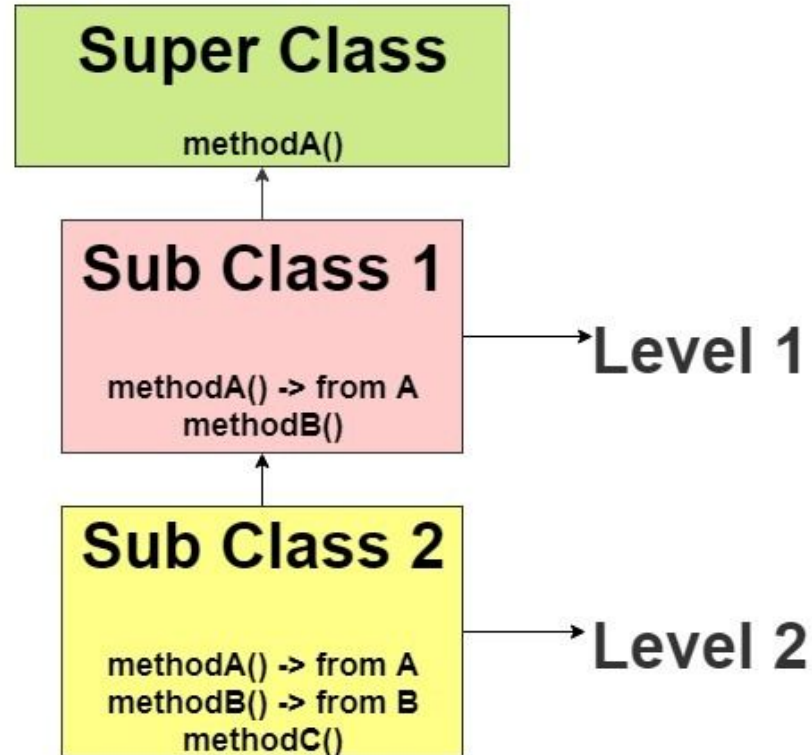
ins = Leader('Rahul', 882016, 'Assistant Manager', 75000, 560)
```

# Multiple inheritance Example



# Multilevel inheritance Structure

## Multi-Level Inheritance



# Multilevel inheritance Example

```
class Base(object):
```

```
    # Constructor
```

```
    def __init__(self, name):
        self.name = name
```

```
    # To get name
```

```
    def getName(self):
        return self.name
```

```
# Inherited or Sub class (Note Person
```

```
class Child(Base):
```

```
    # Constructor
```

```
    def __init__(self, name, age):
        Base.__init__(self, name)
        self.age = age
```

```
    # To get name
```

```
    def getAge(self):
        return self.age
```

```
# Inherited or Sub class (Note Person in bracket)
```

```
class GrandChild(Child):
```

```
    # Constructor
```

```
    def __init__(self, name, age, address):
        Child.__init__(self, name, age)
        self.address = address
```

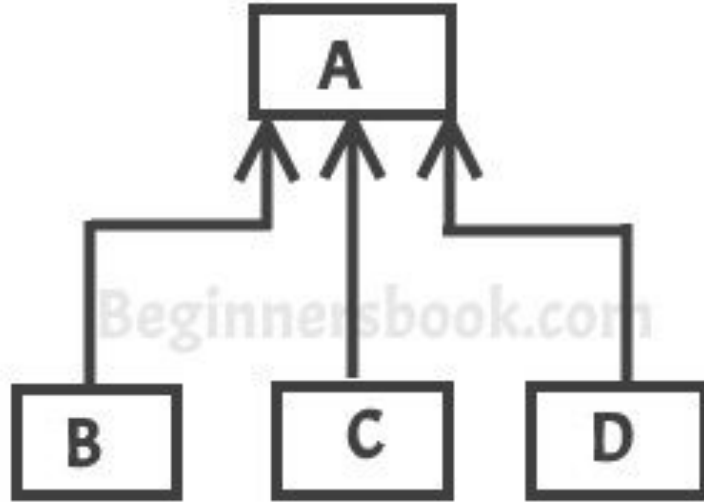
```
    # To get address
```

```
    def getAddress(self):
        return self.address
```

```
# Driver code
```

```
g = GrandChild("Geek1", 23, "Noida")
print(g.getName(), g.getAge(), g.getAddress())
```

# Hierarchical Inheritance Structure

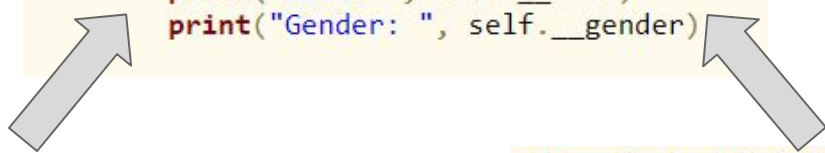


**Hierarchical Inheritance**



# Hierarchical Inheritance Example

```
class Details:
    def __init__(self):
        self.__id="<No Id>"
        self.__name="<No Name>"
        self.__gender="<No Gender>"
    def setData(self,id,name,gender):
        self.__id=id
        self.__name=name
        self.__gender=gender
    def showData(self):
        print("Id: ",self.__id)
        print("Name: ", self.__name)
        print("Gender: ", self.__gender)
```



```
class Employee(Details): #Inheritance
    def __init__(self):
        self.__company="<No Company>"
        self.__dept="<No Dept>"
    def setEmployee(self,id,name,gender,comp,dept):
        self.setData(id,name,gender)
        self.__company=comp
        self.__dept=dept
    def showEmployee(self):
        self.showData()
        print("Company: ", self.__company)
        print("Department: ", self.__dept)
```

```
class Doctor(Details): #Inheritance
    def __init__(self):
        self.__hospital="<No Hospital>"
        self.__dept="<No Dept>"
    def setEmployee(self,id,name,gender,hos,dept):
        self.setData(id,name,gender)
        self.__hospital=hos
        self.__dept=dept
    def showEmployee(self):
        self.showData()
        print("Hospital: ", self.__hospital)
        print("Department: ", self.__dept)
```

# Hierarchical Inheritance Example

```
def main():  
    print("Employee Object")  
    e=Employee()  
    e.setEmployee(1,"Prem Sharma","Male","gmr","excavation")  
    e.showEmployee()  
    print("\nDoctor Object")  
    d = Doctor()  
    d.setEmployee(1, "pankaj", "male", "aiims", "eyes")  
    d.showEmployee()  
  
if __name__=="__main__":  
    main()
```

## Output

```
Employee Object  
Id: 1  
Name: Prem Sharma  
Gender: Male  
Company: gmr  
Department: excavation
```

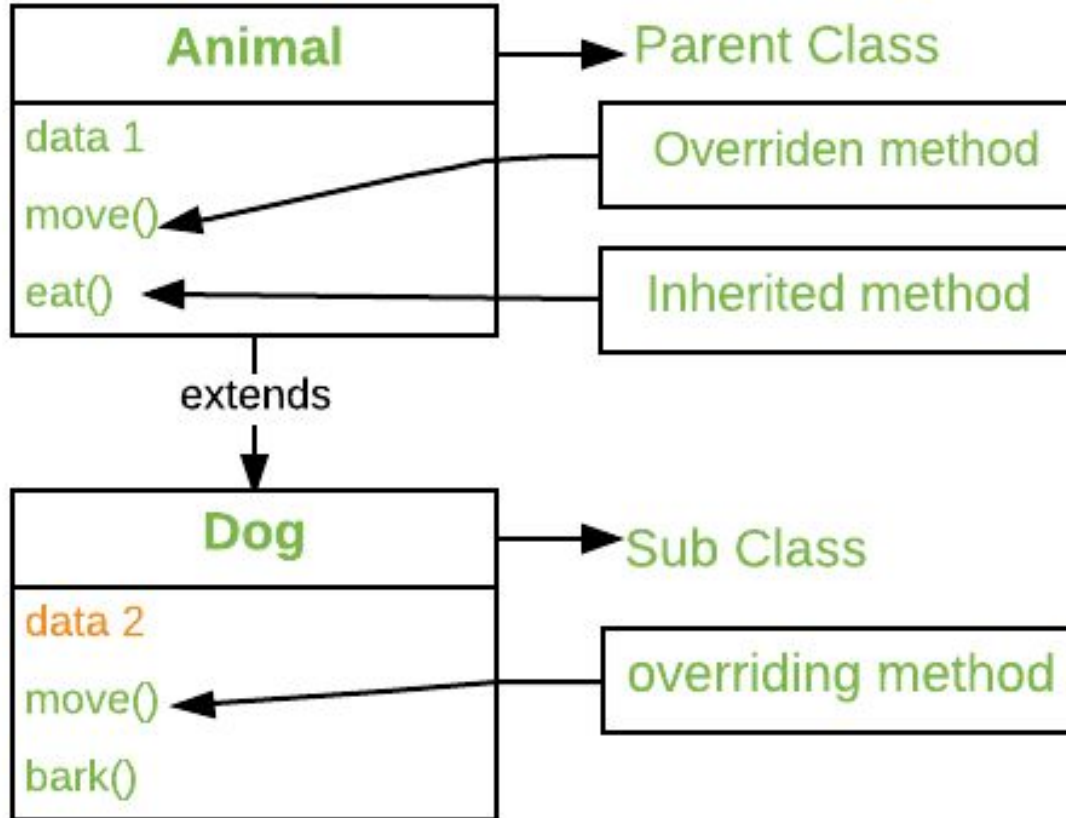
```
Doctor Object  
Id: 1  
Name: pankaj  
Gender: male  
Hospital: aiims  
Department: eyes
```

# Overriding Methods Definition

In Python method overriding occurs simply defining in the child class a method with the same name of a method in the parent class. When you define a method in the **object** you make the latter able to satisfy that method call, so the implementations of its ancestors do not come in play.



# Overriding Methods Example



# Overriding Methods Example

```
# Base Class
class A(object):
    def __init__(self):
        constant1 = 1
    def method1(self):
        print('method1 of class A')

class B(A):
    def __init__(self):
        constant2 = 2
        self.calling1()
        A.__init__(self)
    def method1(self):
        print('method1 of class B')
    def calling1(self):
        self.method1()
        A.method1(self)

b = B()
```

Output:

```
method1 of class B
method1 of class A
```

# Overriding Methods Example

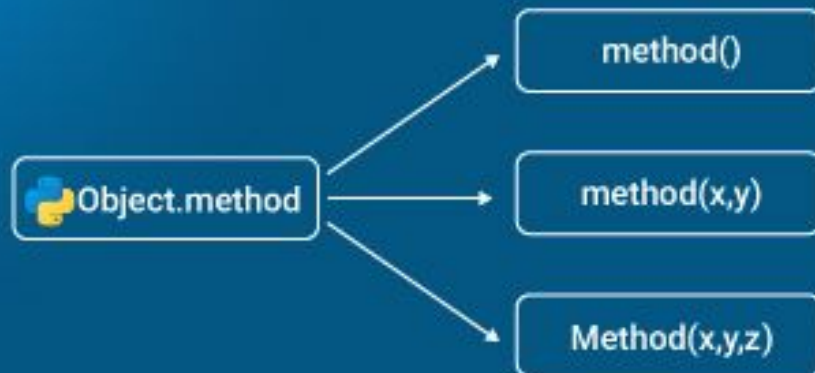
```
class A(object):
    def function1(self):
        print 'function of class A'

class B(A):
    def function1(self):
        print 'function of class B'
        super(B, self).function1()

class C(B):
    def function1(self):
        print 'function of class C'
        super(C, self).function1()

j = C()
j.function1()
```

# Method Overloading in Python





# Methods Overloading

```
# First product method.  
# Takes two argument and print their  
# product  
def product(a, b):  
    p = a * b  
    print(p)
```

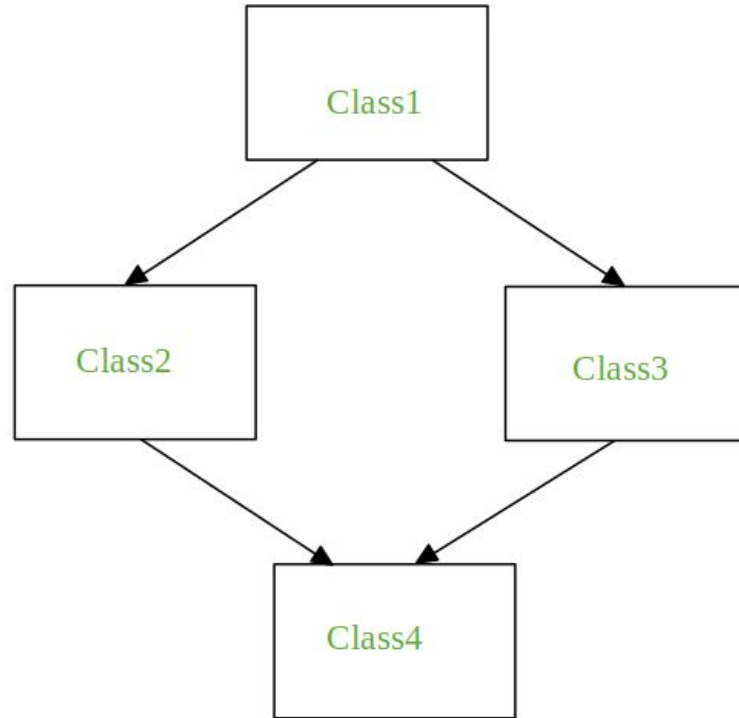
```
# Second product method  
# Takes three argument and print their  
# product  
def product(a, b, c):  
    p = a * b*c  
    print(p)
```

```
# Uncommenting the below line shows an error  
# product(4, 5)
```

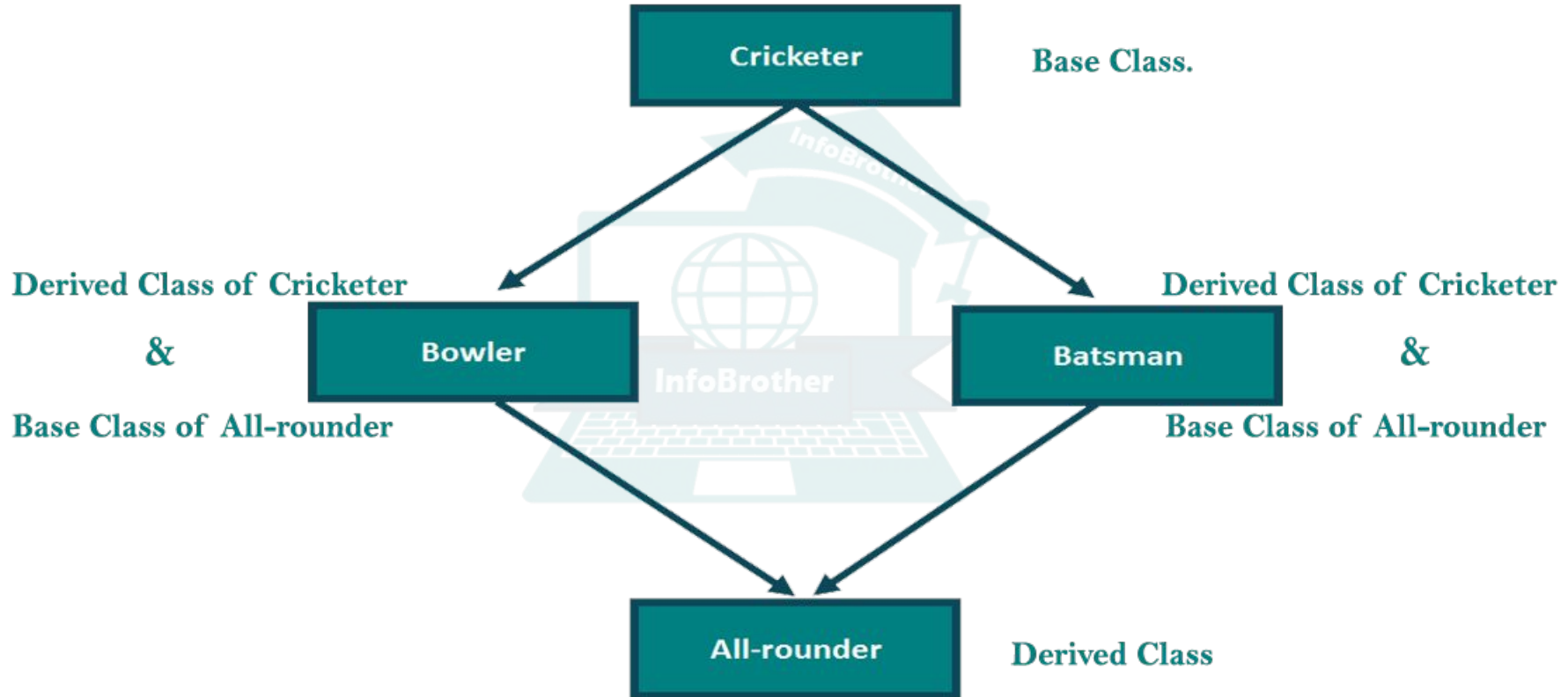
```
# This line will call the second product method  
product(4, 5, 5)
```



# Hybrid Inheritance Structure

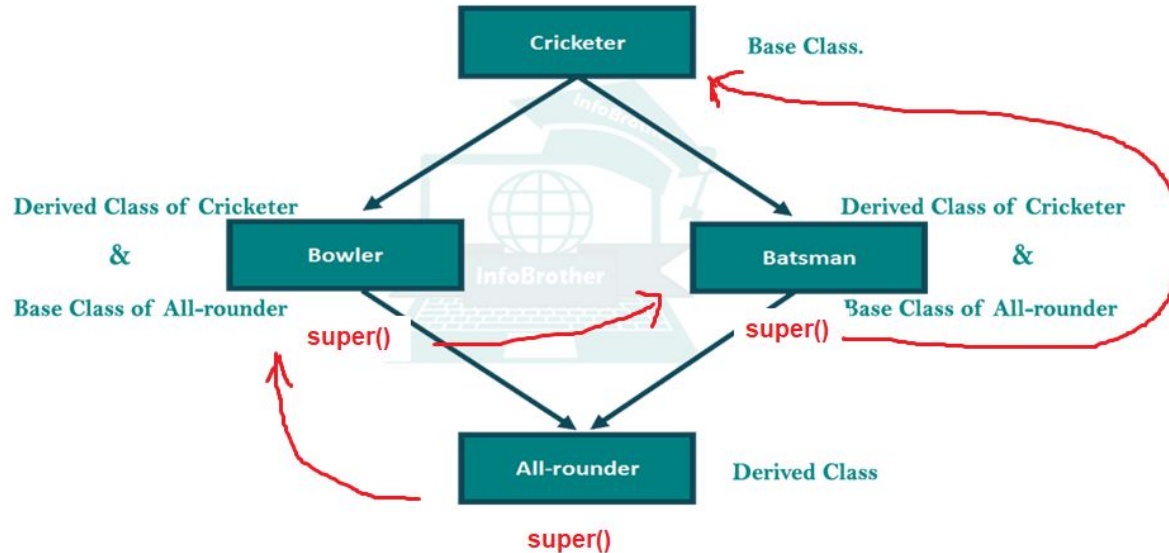


# Hybrid Inheritance Example



# super() method and inheritance

- In a general sense, we know that by super() method we can actually call the parent class. But actually super() method can actually return a sibling class.
- For example, notice the example given below-



# super() method and inheritance

- To understand how `super()` works we have to understand Method Resolution Order (MRO). It is the order in which methods should be inherited in the presence of multiple inheritance. You can view the MRO by using the `__mro__` attribute.

Here is how MRO works:

- A method in the derived class is always called before the method of the base class..
- If there are multiple parents having methods with same name, the method of the parent that appears first will invoke first.

- For this code MRO will be as below -

The Code -

Output-

```
class Cricketer:
    pass

class Bowler(Cricketer):
    pass

class Batsman(Cricketer):
    pass

class Allrounder(Bowler, Batsman):
    pass

print(Allrounder.__mro__)
```

```
(<class '__main__.Allrounder'>,
<class '__main__.Bowler'>,
<class '__main__.Batsman'>,
<class '__main__.Cricketer'>,
<class 'object'>)
```

Our super() method will also follow this order instead of simply returning the base class only.

- As `super()` method follows the MRO, guess the output of the following code-

```
class Cricketer:
    def __init__(self):
        pass

    def printName(self):
        print("I am Cricketer")

class Bowler(Cricketer):
    def __init__(self):
        super().printName()
        super().__init__()

    def printName(self):
        print("I am Bowler")

class Batsman(Cricketer):
    def __init__(self):
        super().printName()
        super().__init__()

    def printName(self):
        print("I am Batsman")
```

```
class Allrounder(Bowler, Batsman):
    def __init__(self):
        super().printName()
        super().__init__()

    def printName(self):
        print("I am Allrounder")

# print(Allrounder.__mro__)
sakib = Allrounder()
```

Assumed Output-

I am Bowler  
I am Cricketer

Actual output-

I am Bowler  
I am Batsman  
I am Cricketer

# Case 1

```
class A:
    def m(self):
        print("In A")

class B(A):
    def m(self):
        print("In B")

class C(A):
    def m(self):
        print("In C")

class D(B,C):
    pass

obj = D()
obj.m()
```

## Case 2

```
class A:  
    def m(self):  
        print("In A")
```

```
class B(A):  
    pass
```

```
class C(A):  
    def m(self):  
        print("In C")
```

```
class D(B,C):  
    pass
```

```
obj = D()  
obj.m()
```



## Case 3

```
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")

class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class2, Class3):
    def m(self):
        print("In Class4")
```

```
obj = Class4()
obj.m()
```

```
Class2.m(obj)
Class3.m(obj)
Class1.m(obj)
```

## Case 4

```
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")

class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class2, Class3):
    def m(self):
        print("In Class4")
        Class2.m(self)
        Class3.m(self)
        Class1.m(self)

obj = Class4()
obj.m()
```

## Case 5

```
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")
        super().m()

class Class3(Class1):
    def m(self):
        print("In Class3")
        super().m()

class Class4(Class2, Class3):
    def m(self):
        print("In Class4")
        super().m()

obj = Class4()
obj.m()
```

## Which type of Inheritance??

```
class Parent:
    def func1(self):
        print("this is function one")
class Child(Parent):
    def func2(self):
        print(" this is function 2 ")
ob = Child()
ob.func1()
ob.func2()
```

---

## Which type of Inheritance??

---

```
class Parent:
    def func1(self):
        print("this is function 1")
class Parent2:
    def func2(self):
        print("this is function 2")
class Child(Parent , Parent2):
    def func3(self):
        print("this is function 3")
```

```
ob = Child()
ob.func1()
ob.func2()
ob.func3()
```

---

# Which type of Inheritance??

```
1  class Parent:
2      def func1(self):
3          print("This is function 1")
4
5  class Child(Parent):
6      def func2(self):
7          print("This is function 2")
8
9  class Child2(Child):
10     def func3(self):
11         print("this is function 3")
12
13  ob = Child2()
14  ob.func1()
15  ob.func2()
16  ob.func3()
```

## Which type of Inheritance??

```
class Parent:
    def func1(self):
        print("this is function 1")
class Child(Parent):
    def func2(self):
        print("this is function 2")
class Child2(Parent):
    def func3(self):
        print("this is function 3")
```

```
ob = Child()
ob1 = Child2()
ob.func1()
ob.func2()
```

---

# Which type of Inheritance??

```
class Parent:
    def func1(self):
        print("this is function one")

class Child(Parent):
    def func2(self):
        print("this is function 2")

class Child1(Parent):
    def func3(self):
        print(" this is function 3"):

class Child3(Parent , Child1):
    def func4(self):
        print(" this is function 4")

ob = Child3()
ob.func1()
```

---





# Thanks to the References

1. <https://www.geeksforgeeks.org/object-oriented-programming-in-python-set-2-data-hiding-and-object-printing/>
2. <https://www.edureka.co/blog/object-oriented-programming-python/>
3. [https://www.w3schools.com/python/python\\_inheritance.asp](https://www.w3schools.com/python/python_inheritance.asp)
4. <https://realpython.com/inheritance-composition-python/>
5. <https://www.programiz.com/python-programming/inheritance>
6. <https://www.edureka.co/blog/inheritance-in-python/>
7. <https://data-flair.training/blogs/python-inheritance/>
8. <https://techvidvan.com/tutorials/python-inheritance/>

Thank you

QA