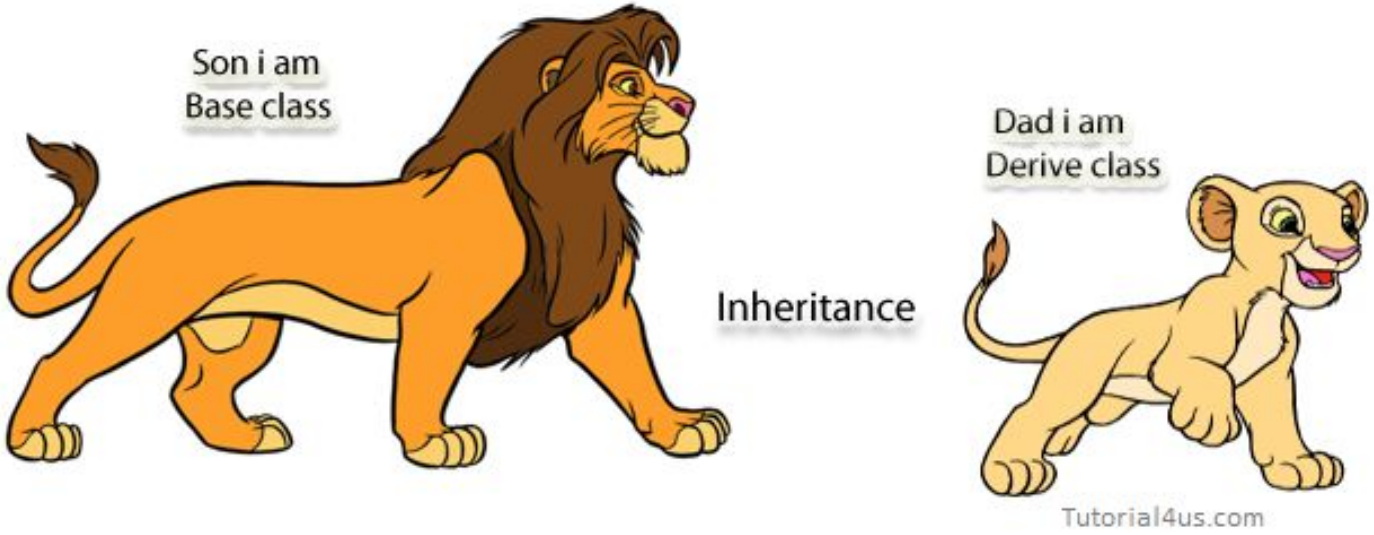
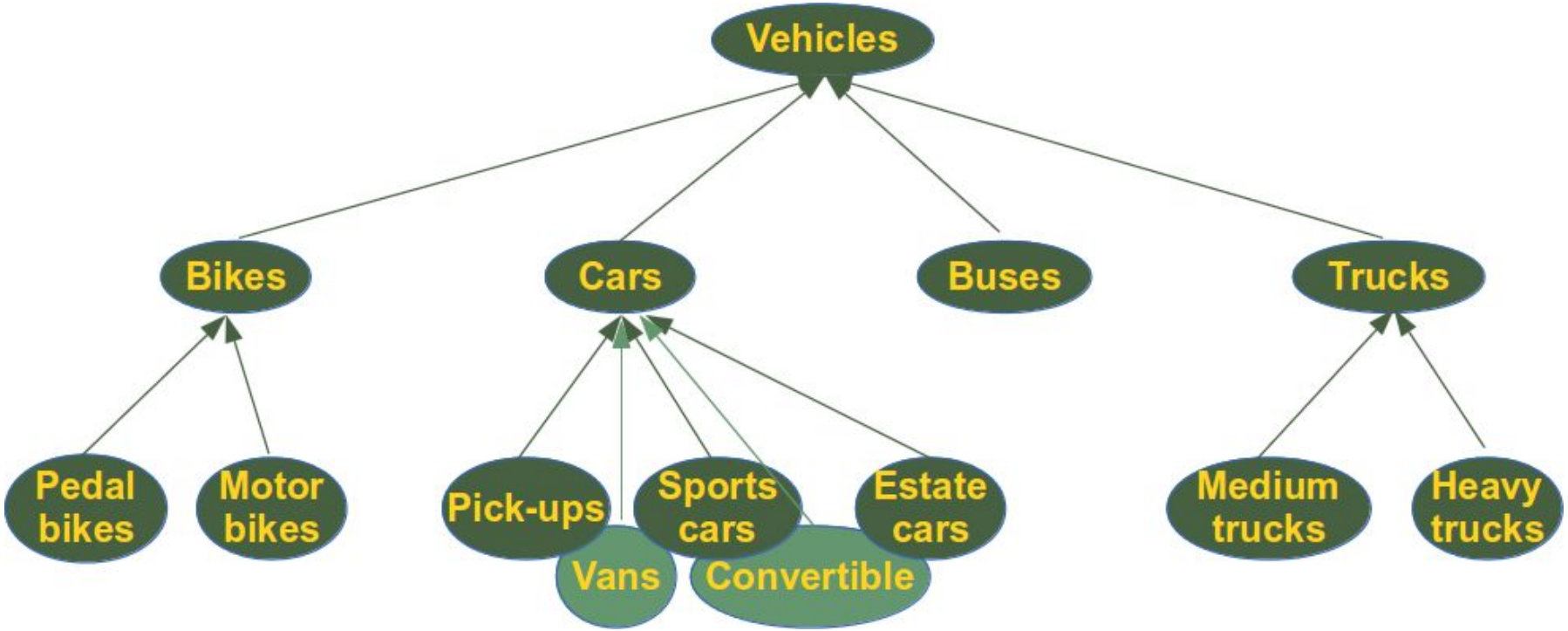
Inheritance

Tanmoy Sarkar Pias



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.

Python

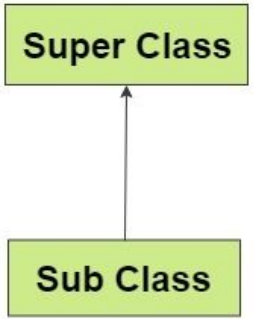
Inheritance

Data Flair

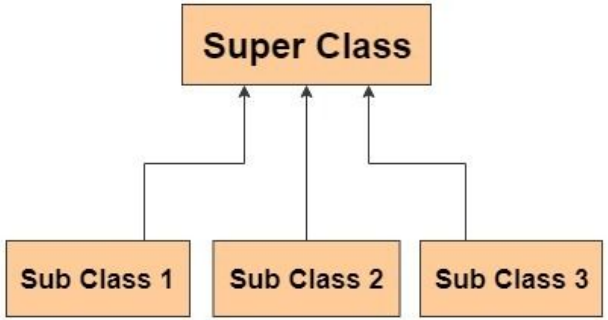
1. Syntax
2. Types of Inheritance
3. The super() function
4. Method Overriding
5. Method Overloading

Types Of Inheritance

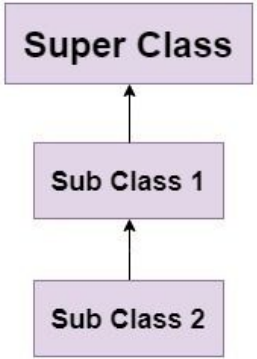
Single Inheritance



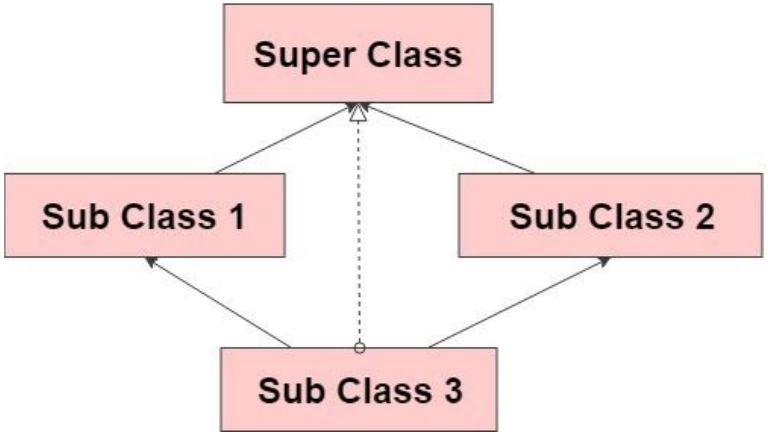
Hierarchial Inheritance



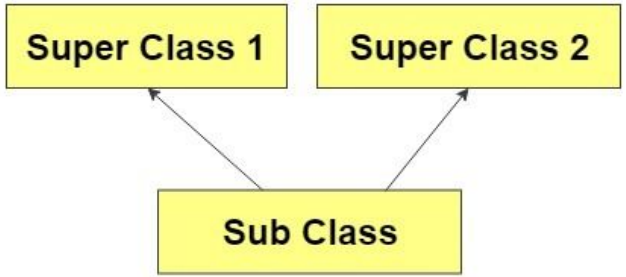
MultiLevel Inheritance



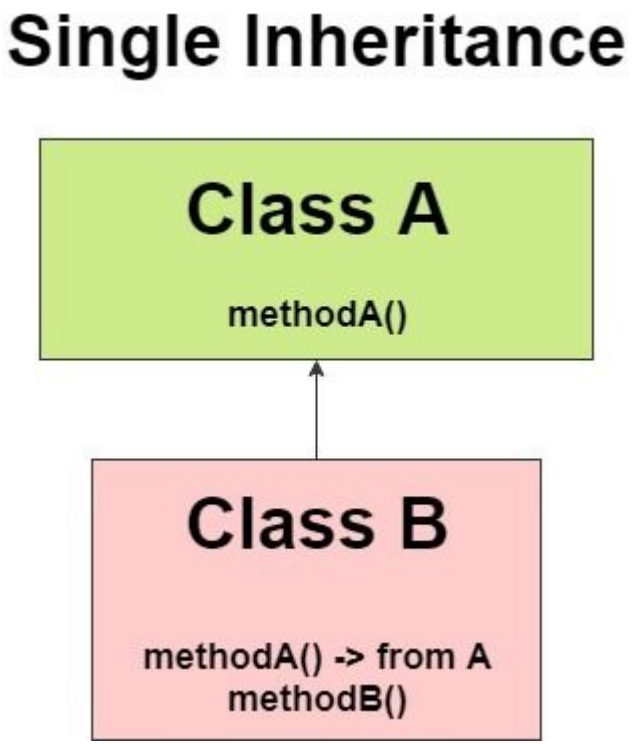
Hybrid Inheritance



Multiple Inhertance



Single Inheritance Structure



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:

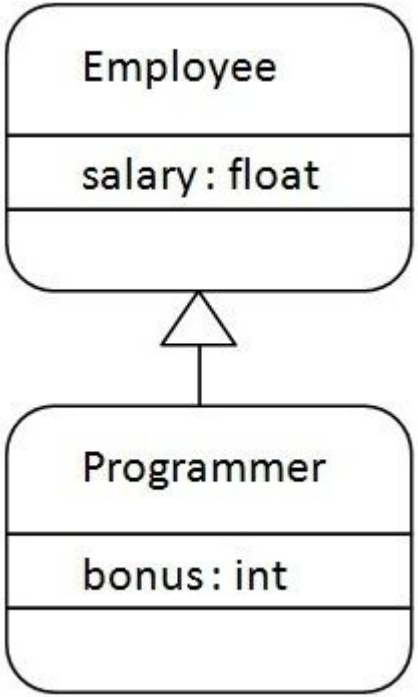
```
class Student(Person):
  pass
```

Example

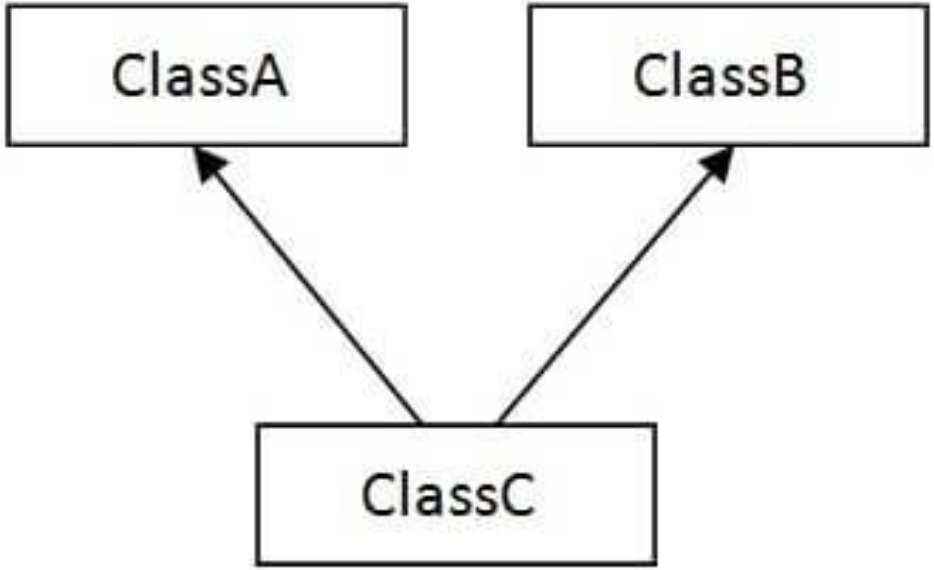
Use the `Student` class to create an object, and then execute the `printname` method:

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

Single Inheritance Example



Multiple inheritance **Structure**



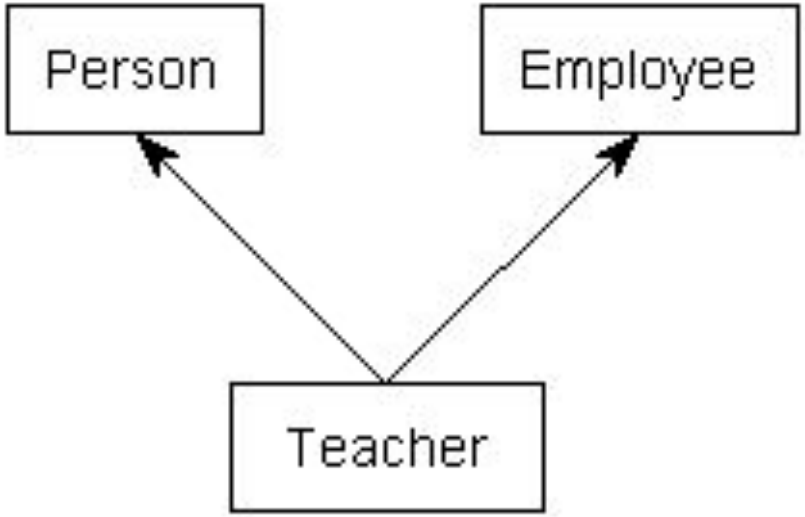
Multiple inheritance Example

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

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

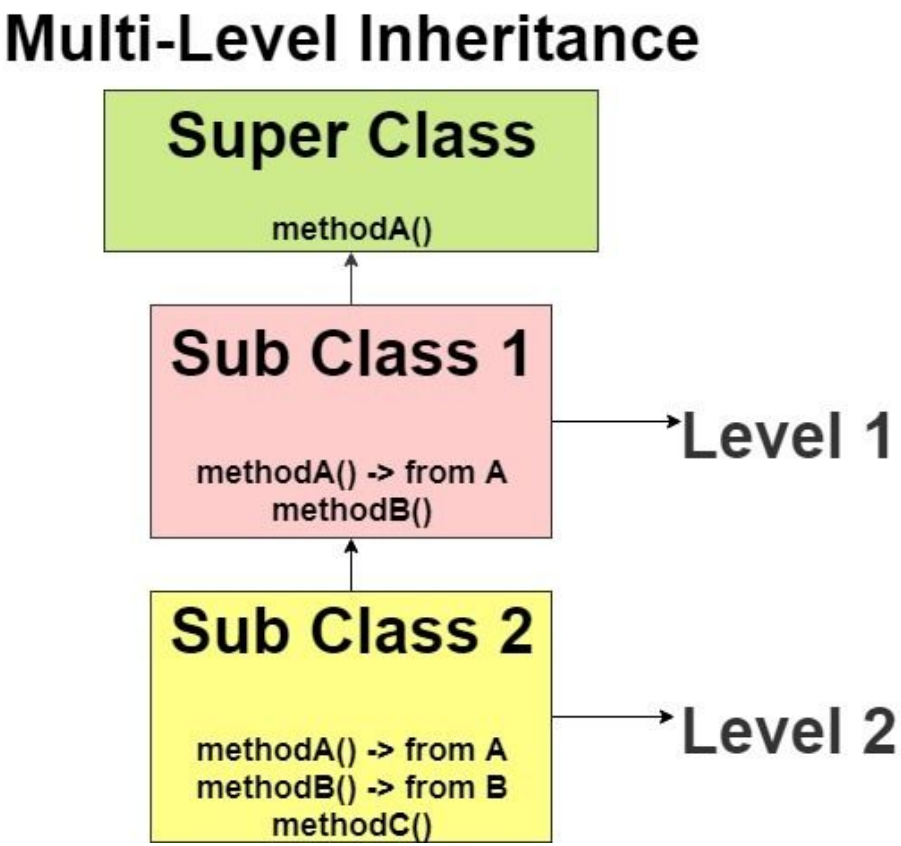
```
class Derived(Base1, Base2):
    def __init__(self):

        # 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()
```

```python
# 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**

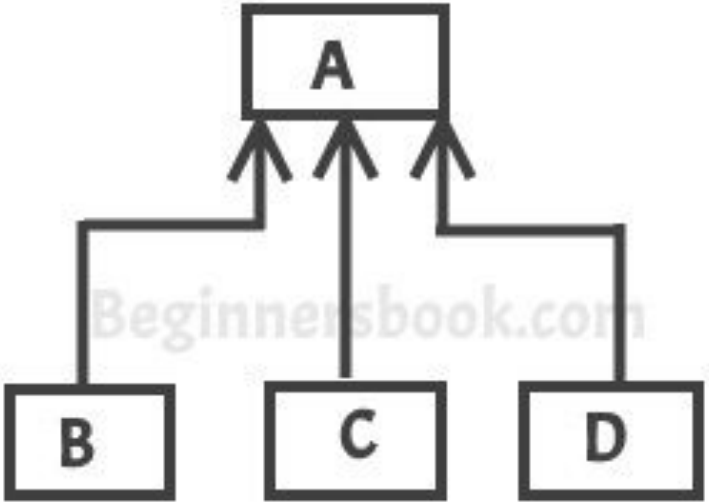


Multilevel inheritance Example

```python
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
```

```python
# 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

```python
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)
```

```python
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)
```

```python
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

```python
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()
```

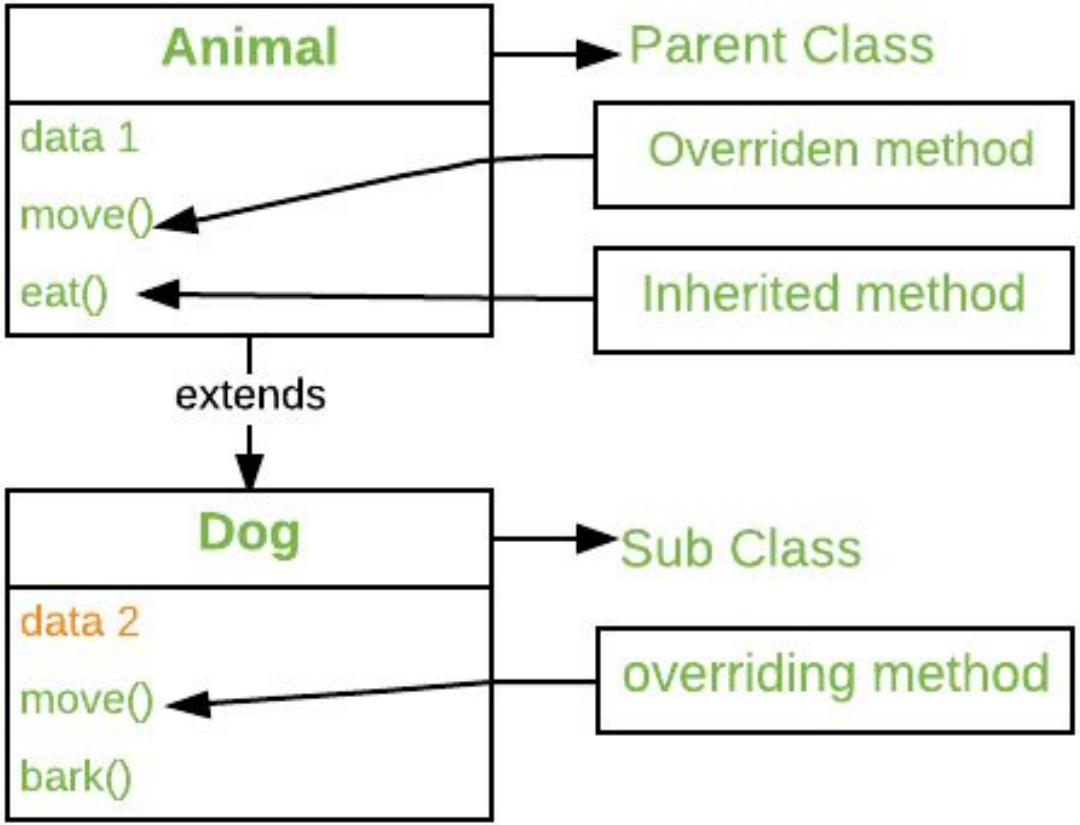
Output

```
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:

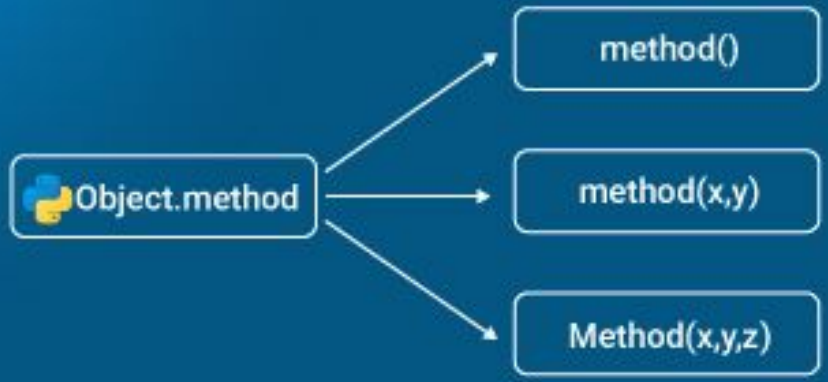
```
method1 of class B
method1 of class A
```

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()
```

edureka!

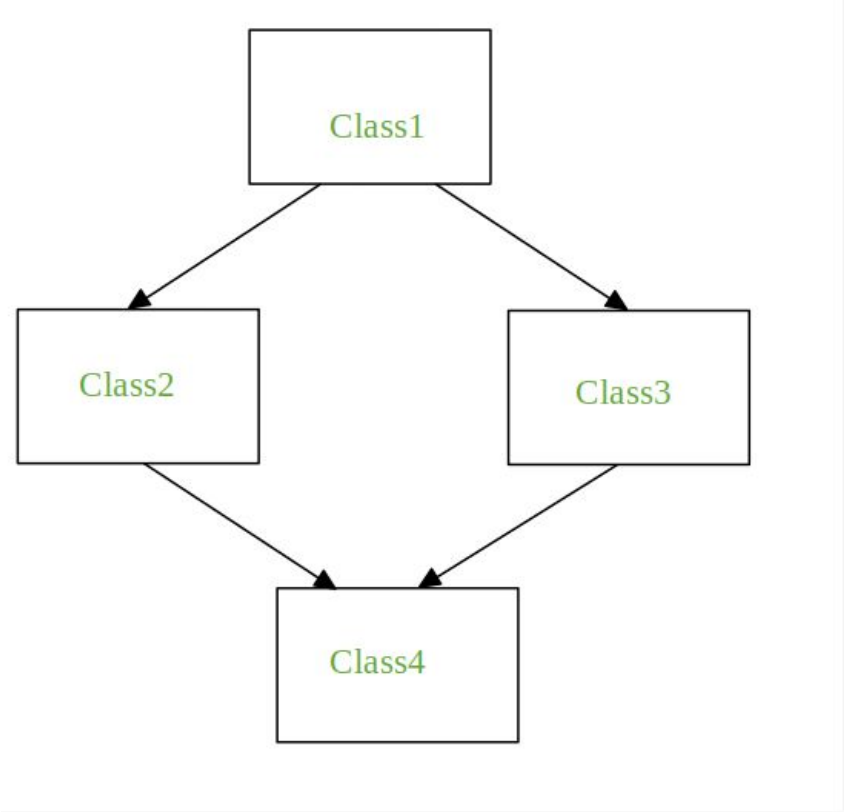
Method Overloading in Python



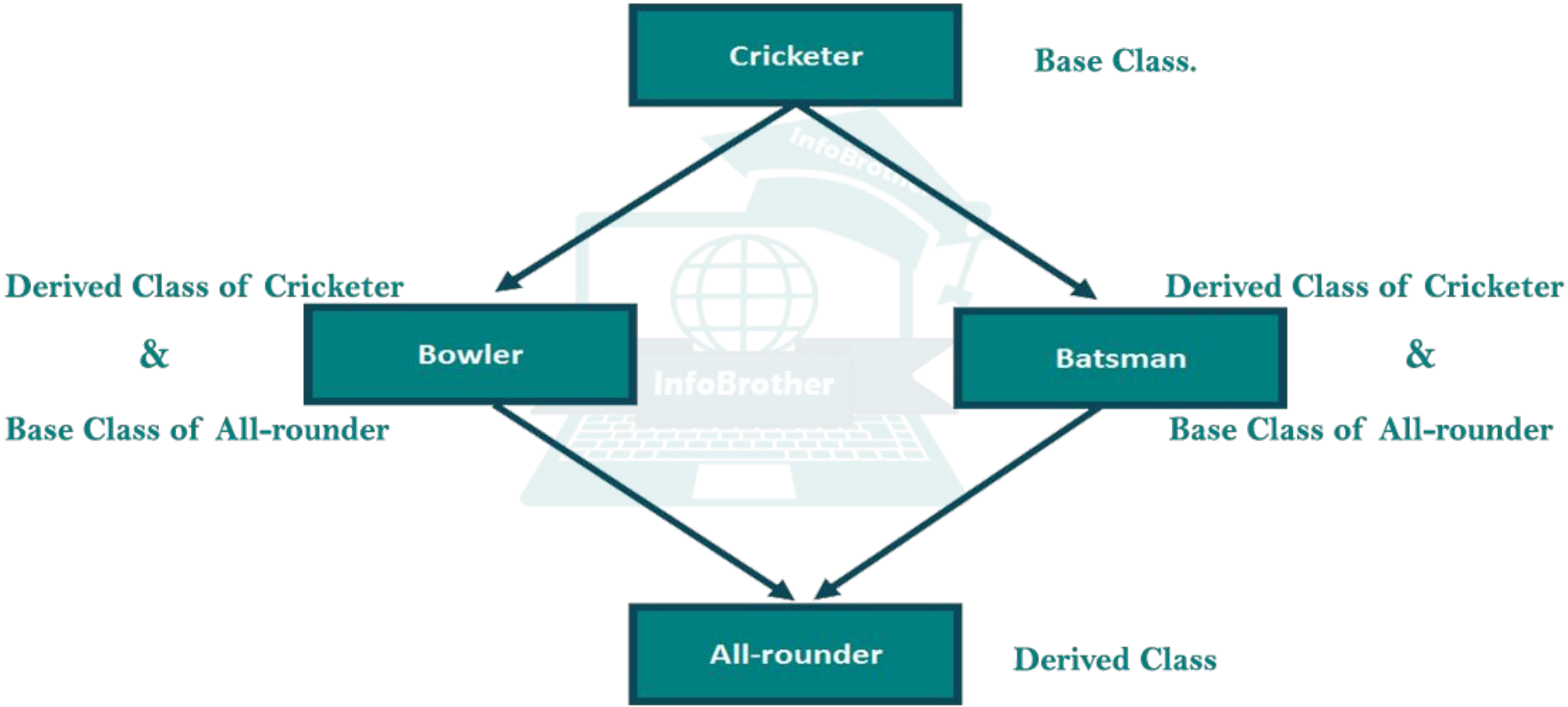
Methods Overloading

```
# 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



Case 1

```python
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):
    pass

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

Case 2

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

class Class2(Class1):
    pass

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

class Class4(Class2, Class3):
    pass

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

Case 3

```python
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")
```

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

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

Case 4

```python
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

```python
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??

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

Which type of Inheritance??

```
class Parent:
    def func1(self):
        print("this is function 1")
class Child(Parent):
    def func2(self):
        Super().func1()
        print("this is function 2")

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

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

class Child(Parent):
    def func1(self):
        print("this is child function")

ob = Child()
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
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