Class Relationships

- Aggregation
- Inheritance

Aggregation(Has-A relationship)

```
# example
class Customer:

  def __init__(self,name,gender,address):
    self.name = name
    self.gender = gender
    self.address = address

  def print_address(self):
    print(self.address._Address__city,self.address.pin,self.address.state)

  def edit_profile(self,new_name,new_city,new_pin,new_state):
    self.name = new_name
    self.address.edit_address(new_city,new_pin,new_state)

class Address:

  def __init__(self,city,pin,state):
      self.__city = city
      self.pin = pin
      self.state = state

  def get_city(self):
    return self.__city

  def edit_address(self,new_city,new_pin,new_state):
    self.__city = new_city
    self.pin = new_pin
    self.state = new_state

add1 = Address('gurgaon',122011,'haryana')
cust = Customer('nitish','male',add1)

cust.print_address()

cust.edit_profile('ankit','mumbai',111111,'maharastra')
cust.print_address()
# method example
# what about private attribute
```

```
gurgaon 122011 haryana
mumbai 111111 maharastra
```

Aggregation class diagram

Inheritance

- What is inheritance
- Example
- What gets inherited?

```
# Inheritance and it's benefits


# Example

# parent
class User:

  def __init__(self):
    self.name = 'nitish'
    self.gender = 'male'

  def login(self):
    print('login')

# child
class Student(User):

  def __init__(self):
    self.rollno = 100

  def enroll(self):
    print('enroll into the course')

u = User()
s = Student()

print(s.name)
s.login()
s.enroll()
```

```
nitish
login
enroll into the course
```

```
# Class diagram
```

What gets inherited?

- Constructor
- Non Private Attributes
- Non Private Methods

```
# constructor example

class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    pass

s=SmartPhone(20000, "Apple", 13)
s.buy()
```

```
 Inside phone constructor
 Buying a phone
```

```
# constructor example 2

class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

class SmartPhone(Phone):
    def __init__(self, os, ram):
        self.os = os
        self.ram = ram
        print ("Inside SmartPhone constructor")

s=SmartPhone("Android", 2)
s.brand
```

```
Inside SmartPhone constructor
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-27-fff5c9f9674f> in <module>
     15
     16 s=SmartPhone("Android", 2)
---> 17 s.brand

AttributeError: 'SmartPhone' object has no attribute 'brand'
```

SEARCH STACK OVERFLOW

```
# child can't access private members of the class

class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    #getter
    def show(self):
        print (self.__price)

class SmartPhone(Phone):
    def check(self):
        print(self.__price)

s=SmartPhone(20000, "Apple", 13)
s.show()
```

```
Inside phone constructor
20000
```

```
class Parent:

    def __init__(self,num):
        self.__num=num

    def get_num(self):
        return self.__num

class Child(Parent):

    def show(self):
        print("This is in child class")

son=Child(100)
print(son.get_num())
son.show()
```

```
100
This is in child class
```

```
class Parent:

    def __init__(self,num):
        self.__num=num

    def get_num(self):
        return self.__num

class Child(Parent):

    def __init__(self,val,num):
        self.__val=val

    def get_val(self):
        return self.__val

son=Child(100,10)
print("Parent: Num:",son.get_num())
print("Child: Val:",son.get_val())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-35-5a17300f6fc7> in <module>
     16
     17 son=Child(100,10)
---> 18 print("Parent: Num:",son.get_num())
     19 print("Child: Val:",son.get_val())

<ipython-input-35-5a17300f6fc7> in get_num(self)
      5
      6     def get_num(self):
----> 7         return self.__num
      8
      9 class Child(Parent):

AttributeError: 'Child' object has no attribute '_Parent__num'
```

SEARCH STACK OVERFLOW

```
class A:
    def __init__(self):
        self.var1=100

    def display1(self,var1):
        print("class A :", self.var1)
class B(A):
```

```
    def display2(self,var1):
        print("class B :", self.var1)

obj=B()
obj.display1(200)
```

```
 class A : 200
```

```
# Method Overriding
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    def buy(self):
        print ("Buying a smartphone")

s=SmartPhone(20000, "Apple", 13)

s.buy()
```

```
 Inside phone constructor
 Buying a smartphone
```

Super Keyword

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    def buy(self):
        print ("Buying a smartphone")
        # syntax to call parent ka buy method
        super().buy()

s=SmartPhone(20000, "Apple", 13)
```

```
s.buy()
```

```
 Inside phone constructor
 Buying a smartphone
 Buying a phone
```

```
# using super outside the class
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    def buy(self):
        print ("Buying a smartphone")
        # syntax to call parent ka buy method
        super().buy()

s=SmartPhone(20000, "Apple", 13)

s.buy()
```

```
 Inside phone constructor
 ---------------------------------------------------------------------------
 RuntimeError                              Traceback (most recent call last)
 <ipython-input-42-b20080504d0e> in <module>
      17 s=SmartPhone(20000, "Apple", 13)
      18
 ---> 19 super().buy()

 RuntimeError: super(): no arguments

  SEARCH STACK OVERFLOW
```

```
# can super access parent ka data?
# using super outside the class
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera
```

```
    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    def buy(self):
        print ("Buying a smartphone")
        # syntax to call parent ka buy method
        print(super().brand)

s=SmartPhone(20000, "Apple", 13)

s.buy()
```

```
Inside phone constructor
Buying a smartphone
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-43-87cd65570d46> in <module>
     19 s=SmartPhone(20000, "Apple", 13)
     20
---> 21 s.buy()

<ipython-input-43-87cd65570d46> in buy(self)
     15         print ("Buying a smartphone")
     16         # syntax to call parent ka buy method
---> 17         print(super().brand)
     18
     19 s=SmartPhone(20000, "Apple", 13)

AttributeError: 'super' object has no attribute 'brand'
```

SEARCH STACK OVERFLOW

```
# super -> constuctor
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

class SmartPhone(Phone):
    def __init__(self, price, brand, camera, os, ram):
        print('Inside smartphone constructor')
        super().__init__(price, brand, camera)
        self.os = os
        self.ram = ram
        print ("Inside smartphone constructor")

s=SmartPhone(20000, "Samsung", 12, "Android", 2)
```

```
print(s.os)
print(s.brand)
```

```
Inside smartphone constructor
Inside phone constructor
Inside smartphone constructor
Android
Samsung
```

Inheritance in summary

- A class can inherit from another class.
- Inheritance improves code reuse
- Constructor, attributes, methods get inherited to the child class
- The parent has no access to the child class
- Private properties of parent are not accessible directly in child class
- Child class can override the attributes or methods. This is called method overriding
- super() is an inbuilt function which is used to invoke the parent class methods and constructor

```
class Parent:

    def __init__(self,num):
      self.__num=num

    def get_num(self):
      return self.__num

class Child(Parent):

    def __init__(self,num,val):
      super().__init__(num)
      self.__val=val

    def get_val(self):
      return self.__val

son=Child(100,200)
print(son.get_num())
print(son.get_val())
```

```
100
200
```

```
class Parent:
    def __init__(self):
```

```
        self.num=100

class Child(Parent):

    def __init__(self):
        super().__init__()
        self.var=200

    def show(self):
        print(self.num)
        print(self.var)

son=Child()
son.show()
```

```
 100
 200
```

```
class Parent:
    def __init__(self):
        self.__num=100

    def show(self):
        print("Parent:",self.__num)

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.__var=10

    def show(self):
        print("Child:",self.__var)

obj=Child()
obj.show()
```

```
 Child: 10
```

```
class Parent:
    def __init__(self):
        self.__num=100

    def show(self):
        print("Parent:",self.__num)

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.__var=10

    def show(self):
```

```
        print("Child:",self.__var)

obj=Child()
obj.show()
```

```
 Child: 10
```

Types of Inheritance

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance(Diamond Problem)
- Hybrid Inheritance

```
# single inheritance
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    pass

SmartPhone(1000,"Apple","13px").buy()
```

```
 Inside phone constructor
 Buying a phone
```

```
# multilevel
class Product:
    def review(self):
        print ("Product customer review")

class Phone(Product):
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
```

```
        print ("Buying a phone")

class SmartPhone(Phone):
    pass

s=SmartPhone(20000, "Apple", 12)

s.buy()
s.review()
```

```
 Inside phone constructor
 Buying a phone
 Product customer review
```

```
# Hierarchical
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    pass

class FeaturePhone(Phone):
    pass

SmartPhone(1000,"Apple","13px").buy()
FeaturePhone(10,"Lava","1px").buy()
```

```
 Inside phone constructor
 Buying a phone
 Inside phone constructor
 Buying a phone
```

```
# Multiple
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")
```

```
class Product:
    def review(self):
        print ("Customer review")

class SmartPhone(Phone, Product):
    pass

s=SmartPhone(20000, "Apple", 12)

s.buy()
s.review()
```

```
 Inside phone constructor
 Buying a phone
 Customer review
```

```
# the diamond problem
# https://stackoverflow.com/questions/56361048/what-is-the-diamond-problem-in-python-and-why-
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class Product:
    def buy(self):
        print ("Product buy method")

# Method resolution order
class SmartPhone(Phone,Product):
    pass

s=SmartPhone(20000, "Apple", 12)

s.buy()
```

```
 Inside phone constructor
 Buying a phone
```

```
class A:

    def m1(self):
        return 20

class B(A):
```

```
    def m1(self):
        return 30

    def m2(self):
        return 40

class C(B):

    def m2(self):
        return 20
obj1=A()
obj2=B()
obj3=C()
print(obj1.m1() + obj3.m1()+ obj3.m2())
```

```
 70
```

```
class A:

    def m1(self):
        return 20

class B(A):

    def m1(self):
        val=super().m1()+30
        return val

class C(B):

    def m1(self):
        val=self.m1()+20
        return val
obj=C()
print(obj.m1())
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-56-bb3659d52487> in <module>
     16         return val
     17 obj=C()
---> 18 print(obj.m1())
```

Polymorphism

- Method Overriding
- Method Overloading
- Operator Overloading

```
     16         return val
```

```
class Shape:

  def area(self,a,b=0):
    if b == 0:
      return 3.14*a*a
    else:
      return a*b

s = Shape()

print(s.area(2))
print(s.area(3,4))
```

```
12.56
12
```

```
'hello' + 'world'
```

```
'helloworld'
```

```
4 + 5
```

```
9
```

```
[1,2,3] + [4,5]
```

```
[1, 2, 3, 4, 5]
```

Abstraction

```
# nitish.campusx@gmail.com
```