**OOP**

-------------

Everything is a object

Object is real time entity

Class is property and behavior of object

it is a blue print of a object

attributes: properties

methods: behavior and characteristics

EXAMPLE:

class: GOD

object: SHIVA

attributes: UNSEEN, has incredible powers, human form ,

behavior : does favour for humans (NOTE:except ENGINEERS)

class:Programming LANG

object: Python

attributes: no use of ; and brackets , human understandable

behavior: interpretes line by line

**4 Modules:**

>>encapsulation

packing related things as one

>>polymorphism (poly=many , morph=shapes)

same function but different object for different set of values

>> DATA abstraction

hiding unnecessary variables for user and other classes

>>inheritence

inherit or reuse code from base class or parent class

relation btw the class (parent and child )

**CLASS**

--------------------------

syntax:

class Classname:

stmt

mtds & attr

obj=Classname() <object> //instantiating a class

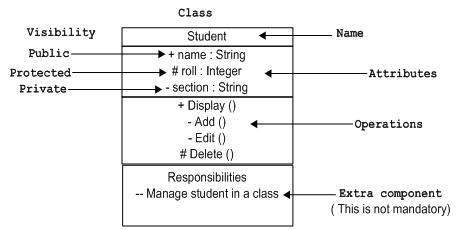
>>>creating a object / instance is called instantiation of that class

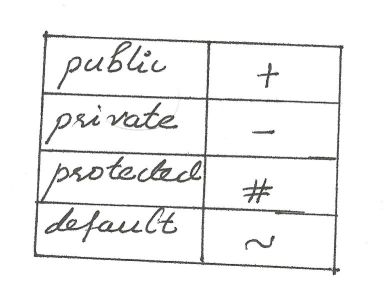
>>>memory will be allocated only when a object is created for a class .

>>>printing a object will give the address(hexa value ) of that instance of a class

>>> UML diagram is a representation of a class in a pictorial form

**UML diagram**





class Dog:

breed='lab'

age='adult'

def run(<self>,a):

print("runnning",a)

obj=Dog()

obj.run(a) <runs when self is mentioned in def>

Dog.run() <runs when there is no self is mentioned because there no obj/instance will created>

**Constructor**

def \_\_init\_\_(self): //Constructor function

pass

A Constructor special method will be invoked when ever a object is created for that particular class

Example:

class Student:

def \_\_init\_\_(self,name,rno,dept,cgpa):

self.name=name //INSTANCE VARIABLE

self.rno=rno

self.dept=dept

self.cgpa=cgpa

def print(self):

print(self.name,self.rno,self.dept,self.cgpa)

def hobby(self):

hby=input("enter hby")

print(hby)

def learn(self):

lrn=input("enter hrs")

print(lrn)

def sleep(self):

slp=input("enter hrs(slp)")

print(slp)

jeevan=Student('raghul','17cse63','cse','6.8')

jeevan.print()

jeevan.hobby()

jeevan.learn()

jeevan.sleep()

**Class variable:**

It is a variable that is shared among the data members of the clas

It is defined within the class but outside any of the class methods

**Instance variable**

A var that is defined and belongs only to the current of the class

Code to change the class variable out side the class

EXAMPLE

Class DOG:

Breed=’lab’ #class variable

obj=DOG()

obj1=DOG()

obj.Breed=”Rodasian ridtchback” #object variable

Print(obj.Breed)

Print(DOG.breed)

We can call the instance variable only with the help of object name not with the class name

**Inbuilt functions across attributes**

**getattr(<objectname>,<attrb\_name>) returns the value of attr**

**setattr(<objectname>,<attr\_name>,<new value>) modifies**

**hasattr(<objectname>,<attrb\_name>) returns true or false**

**delattr(<objectname>,<attrb\_name>) modifies**

**<classname>.\_\_name\_\_**

**<classname>.\_\_doc\_\_**

**<classname>.\_\_class\_\_**

**<classname>.\_\_bases\_\_**

**<classname>.\_\_module\_\_**

**EXAMPLE PROGRAM ###\_\_\_EMPLOYEE\_\_\_###:**

**class Employee:**

**def \_\_init\_\_(s):**

**s.emp\_name=''**

**s.emp\_salary=0**

**s.emp\_ID=''**

**s.emp\_dsgn=''**

**s.emp\_level=''**

**s.Input\_data()**

**def Input\_data(s):**

**s.emp\_name=input("enter name ")**

**s.emp\_ID=input("enter ID ")**

**s.emp\_dsgn=input("enter designation ")**

**s.emp\_salary=int(input("enter salary"))**

**def allocate\_levels(s):**

**if s.emp\_salary>=25000:**

**s.emp\_level='A'**

**elif s.emp\_salary>=20000 and s.emp\_salary<25000:**

**s.emp\_level='B'**

**elif s.emp\_salary>=15000 and s.emp\_salary<20000:**

**s.emp\_level='C'**

**else:**

**s.emp\_level='D'**

**def show(s):**

**for k,v in s.\_\_dict\_\_.items():**

**print(k,":",v)**

**def get\_data(obj):**

**for i in range(len(obj)):**

**print("enter the employee {} details".format(i+1))**

**obj[i]=Employee()**

**def get\_max\_sal(obj):**

**maxx=0**

**for i in range(len(obj)):**

**obj[i].allocate\_levels()**

**obj[i].show()**

**print(end='')**

**if obj[i].emp\_salary>maxx:**

**maxx=obj[i].emp\_salary**

**indx=i**

**print()**

**print("maximum salary is got by ",end='')**

**print(obj[indx].emp\_name)**

**def main():**

**obj=[i for i in range(int(input("enter num of employees")))]**

**get\_data(obj)**

**get\_max\_sal(obj)**

**if \_\_name\_\_=='\_\_main\_\_':**

**main()**

**Destructor**

**----------------**

**syntax**

**def \_\_del\_\_(self):**

**stmts**

**description:**

**del <object> the control will go to \_\_del\_\_ from this statement**

**ENCAPSULATION:**

We can able to restrict the access to variables and methods

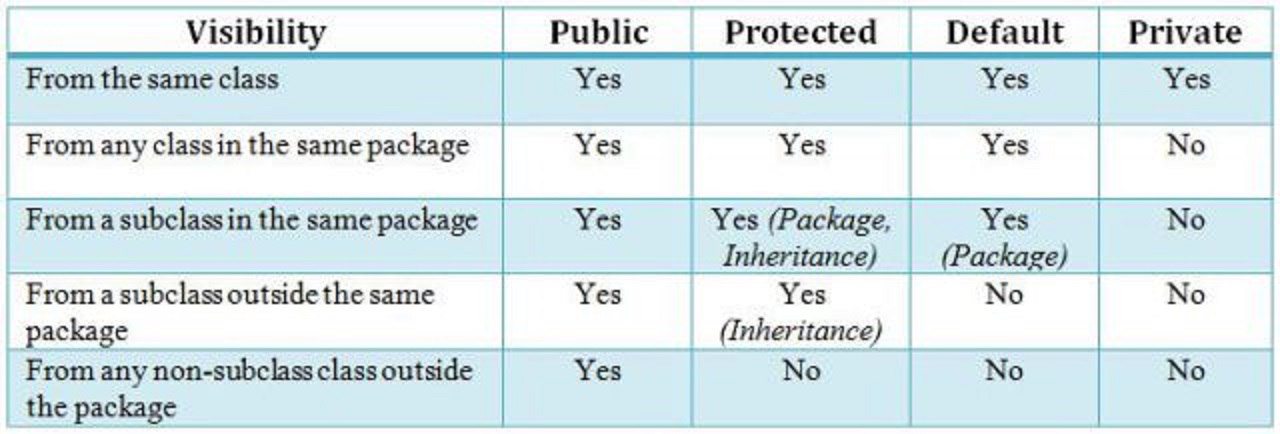
We can prevent the data being modified encapsulation is achieved by access specifiers. It provides data security

Access modifiers UML python code

PUBLIC + <no keywords>

Protected # \_<var name>

Private - \_\_<var name>

**

Inorder to access and modify private variables we have an indirect way called getters and setters