Object Oriented Programming

→ OOP gives a programmar to make his own data type

Class:

→ Object of the class we call variable

L = [1, 2, 3] → built in class

append, pop, insert → methods

What is class ?

→ Class is a blueprint which describes how it's object will behave.

Example → Car is a class, BMW is an object of that class

What is an object ?

→ Object is an instance of a class, which can use the attributes and methods of the class

BMW = Car ()

→ class

→ Object

There can be two types of classes →

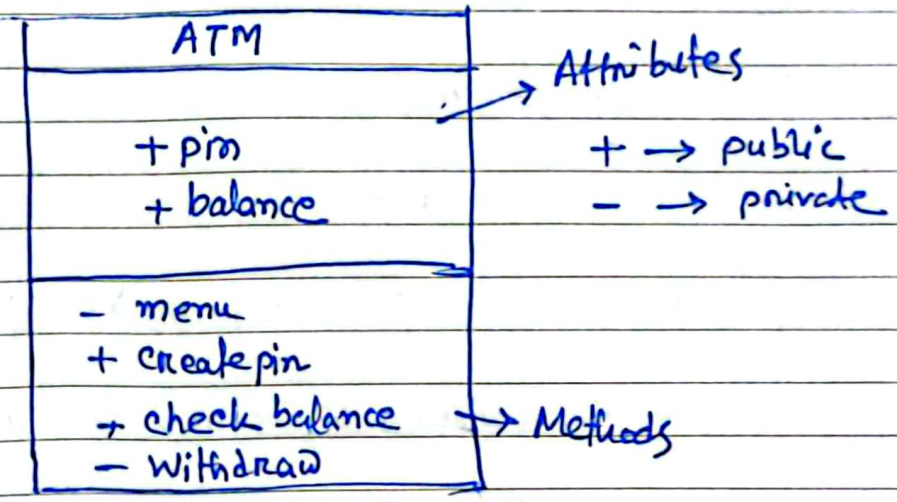
1) Built in classes
2) User defined classes

→ When you create an object of a class, constructor of that class gets executed automatically.

Method vs functions?

→ Those functions who are defined under class, they are methods But those functions who are defined independently outside of a class, they are functions.

Diagram of a Class



What are magic methods →

These are some method who get some extra abilities than the normal methods and they have __x__ like this naming convention

__init__ → Executes automatically when an obj is created

What is the main purpose of a constructor?

→ Constructor is used to initialize and do configuration related code. So that when an object is created, the configuration can get executed immediately.

Notes

What is the purpose of self ?

In a class, everything attributes, & methods belong to the object of the class. Without the object of the class, no method can use any attribute or call any other method inside the class. So to communicate with each other attributes and methods need a common relationship or belongings which is the object itself. and self represent that object. So, in a nutshell no attribute or method can communicate with each other in a class without the help of an object. object always work as the medium for the communication.

Another magic method → `__str__()`:
return .....
When you print your object, the str method provokes.

Another magic method → `__add__(self, other)`
→ it takes two objects and add them

Another → `__sub__(self, other)`, `__mul__(self, other)`

Notes

OOP Part-2

You can declare attributes outside the class, using the object.

What is a reference variable?

if p = Person() some @lass

↳ is called the reference variable of the created object, which contains the memory loc

q = p

↳ Now q is pointing to the same location

* Pass By Reference:

→ We can pass the object of a class as input parameter of a function

→ Similarly a function can also return an object of a class as output.

So, we pass an object to a function or a function returns the object of a class, that's ~~called~~ actually sharing the same memory location of the object. That's call pass by reference

So, objects like class, list, other mutable objects, sets, dictionaries if they go under a function and got modified while returning their memory address will not change but for immutable objects

Notes

• their memory with address will change.

* Encapsulation

What are instance variables?

→ They are such variables whose values are different based on objects to objects.
For every objects, instance variable values are different

Encapsulation is a technique of data security.

Means, suppose the attributes you have added under your class constructor, anyone with the object name can modify it from outside.

So, if you don't prevent access management from outside a serious harm can be done. So a layer of security can be given by encapsulation.

You can make your attributess private by adding dubble underscore. → self.__name = 'Faisal'.

Also you can make your class methods private like the same

example → def __menu()

So, when you make a variable name private, the

ultimate name get changed to → _class name__var name

Notes

So, suppose if some one knows your private variable name

__name, and from outside he tried to change

obj.__name = something else, it ~~won't~~ will create

an attribute in the class but won't change your

private variable value, Because actually to access

your private variable/attribute, the name is →

_Atm__name

Using only this someone can access your private variable

So, suppose someone knows this concept also?

Then he can definitely modify attributes ~~also~~ in the class

though they are private.

obj._Atm__name = somethingelse, will definitely

modify the private attribute of the class from outside

Why in python things are not truely made private?

→ Because, Python is pi made for adults.

5

When you make private variable/attribute, you have to make

setters and setters method ~~from outs~~ to access those

from outside.

/ /

Notes

Everything in python is an object but not all object is mutable

* Static Keyword (Static variable)

Instance variables are variables of an object (Under constructors)

Static variables are variable of a class.

-> You can declare it outside the constructor but inside the class. as normal,

-> But in constructor you have to update is it using class name

instead of self.counter, use Atm.counter

You can also make static variables private by using double _

There are also some methods in which you don't use self,

In that cases, they become class methods. From outside if you want to call them you have to use classname.methodname

instead of object name.methodname()

Notes

* Abstraction → A process of handling complexity by hiding unnecessary information from users.

Example of Abract method in python →

```
from abc import ABC, abstractmethod

class BankApp(ABC):

    def database(self):

        print("Database Connected")

    @abstractmethod

    def security(self):
        pass


Class MobileAPP(BankApp):

    def mobile_login(self):
        print("")

    def security(self):
        print("")
```

→ Have to use this method which is defined as abstract method in parent class, else code won't work.

A class is called an abstract class if it contains an abstract method

Also, We can't create an object of an abstract class.

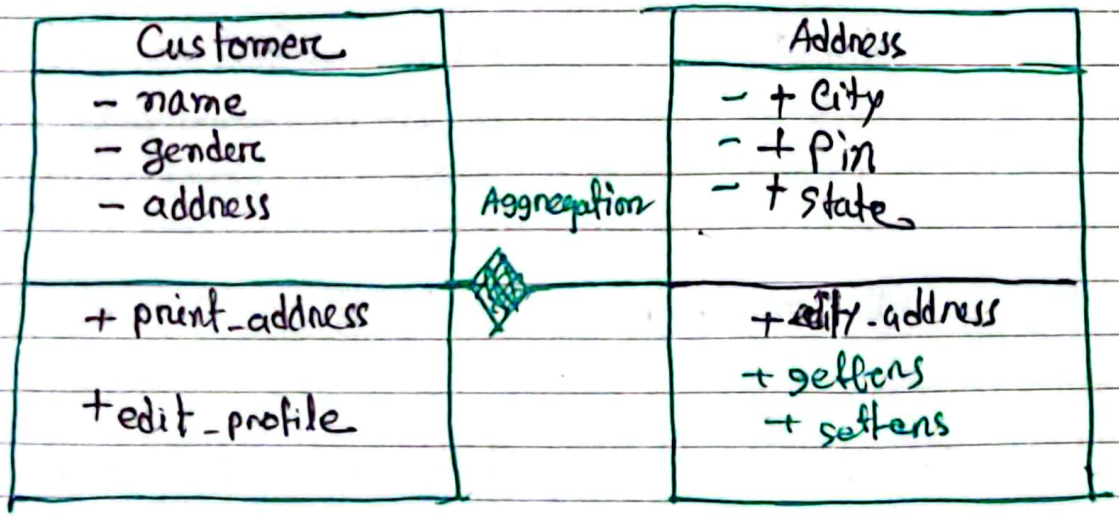
* Class Relationship

Aggregation: (Has a relationship)

Suppose your customer class needs name, gender and address in which address itself is an object of another class which has city, pin, state, province like attributes. So customer class is aggregating the contents of address class. But if some attributes of address class is private, then customer class won't be able to access them. Then you need getters and setters to access.

As customer class has a relationship with address class, it's an example of aggregation.

* Class Diagram of Aggregation



Notes

* Inheritance

What's the biggest advantage of applying Inheritance?

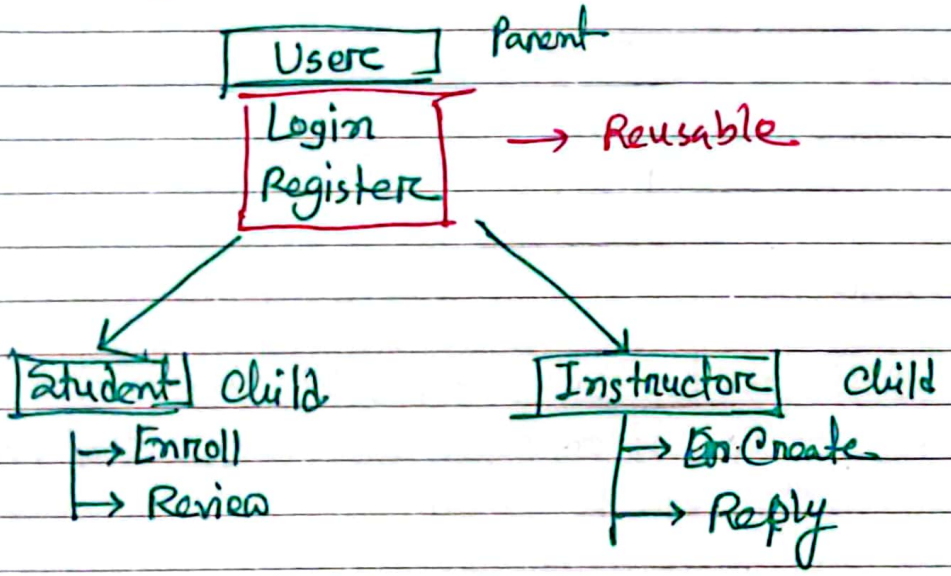
→ Code Reusability

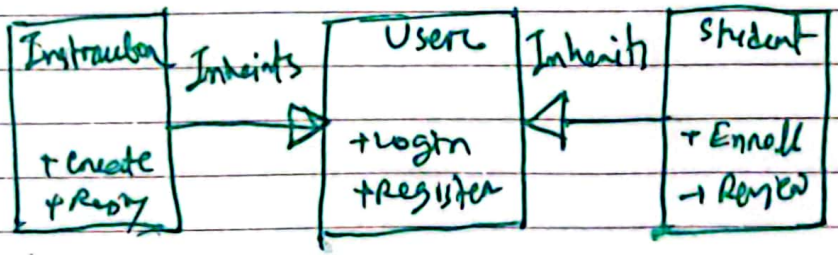
Normal class →

Student	Instructor
Login	Login
Register	Register
Enroll	Create
Review	Reply

→ Repeated codes / logics

After applying inheritance





Notes

* What's get inherited?
  - → Constructor
  - → Non private Attributes
  - → Non private methods

→ If child class has no constructor of it's own, then python go and use parent class constructor and executes it if it is not private else, it will use it own constructor when an object of child constructor.

You can assign values to the parent class while making an object of the child class but you can't access any private attribute or method of your parent class, using child class object.

* in python either it's private or public. No protected.

* Method Overriding:

If child class has same method like parent class, in that case if an object call that method, child class's method will be called.

Notes

* Super Keyword

Suppose you have a same method named buy in both parent and child class. Now you declared an object for the child and using that ~~child~~ object if you call buy, your child class buy will be called. But if you used ~~super.buy~~ super().buy() ~~and~~ under your child class buy method, then parent class ~~not~~ buy method will be called.

You also can do that for constructor, you can call your parent class constructor from your child class constructor

```
class Smartphone(phone):

    def __init__(self, price, brand, camera, os, ram):

        super().__init__(price, brand, camera)
        self.os = os
        self.ram = ram
```

* You can not ~~call using~~ use Super outside of class.

* You can not access attributes using super(), can access constructors and methods only.

Notes

* Inheritance Summary

→ A class can inherit from another class

→ Inherit improves code reusability

→ Constructors, methods, attributes get inherited to the child class

→ The parent has no access to the child class

→ Private properties of parent class is not accessible by child class unless getters and setters are there

→ Child class can override the attributes or method (method overriding)

→ super() → inbuilt function, used to invoke the parent class methods or constructors.

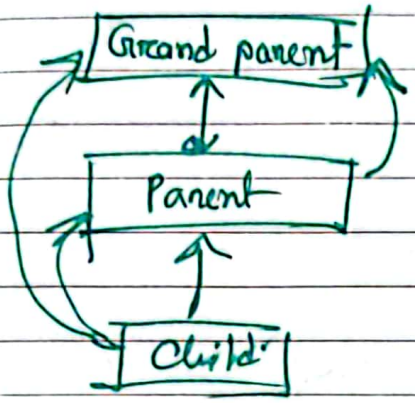
* Types of Inheritance:

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

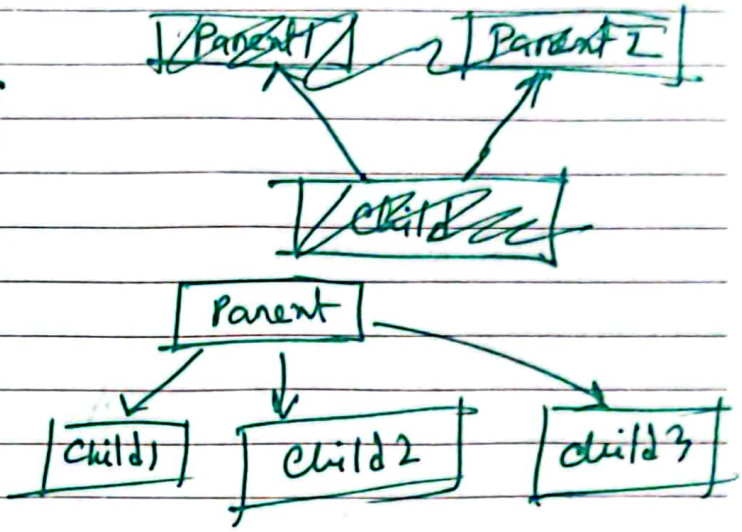
Single

Parent
↑
child

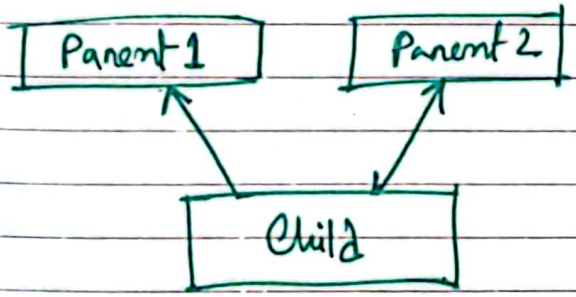
Multilevel



Hierarchical

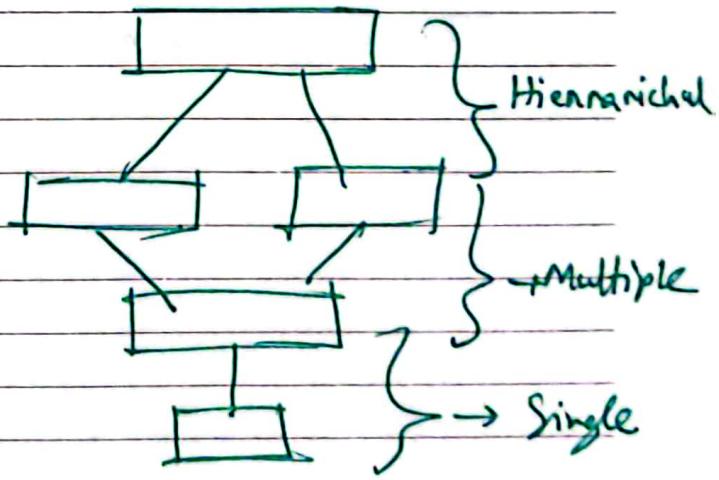


Multiple



→ Diamond problem

Hybrid



Notes

What is diamond problem?

If your class is an example multiple inheritance, when it has more than one parent. Suppose, in both or all the parent class there is a method defined named "buy", so, child class will use whose method?

In python it will use the method of that parent class which it has called first while being created

child class → class child (Parent1, Parent2)

↳ This class's method will be used.

This concept is called → Method resolution order.

Notes

* Polymorphism → Multiple forms

- Method Overriding
- Method Overloading
- Operator Overloading

→ Method overriding has already been discussed.

Method overloading

```
class shape:

    def area(self, rad):
        return 3.14*rad

    def area(self,l,b):
        return l*b;
```

In the same class, same name methods but take different parameters

→ The usability of method overloading is the code is more clean and readable

But python don't support this method overloading concept. It will take the last method as only method if both methods have same name in a class.

So, if you call the first area method by giving rad parameter It won't work.

It will work when you will provide l and b parameter while calling area method.

But Python support operator overloading when we use magic methods (+, -, *, /)

Notes