RANKA
DATE / /
PAGE

Week-3

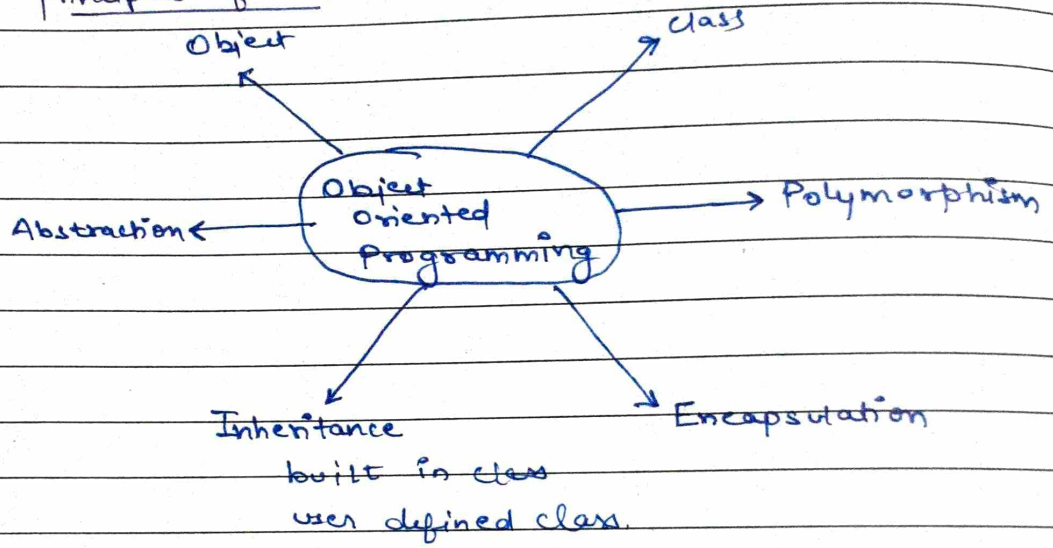
OOP- class-1

L = [1,2,3]
L.upper() → 'List' object has no attribute upper.

OOP- need — (Generality to Specificity)

The OOP gives programmer power to create own datatypes.

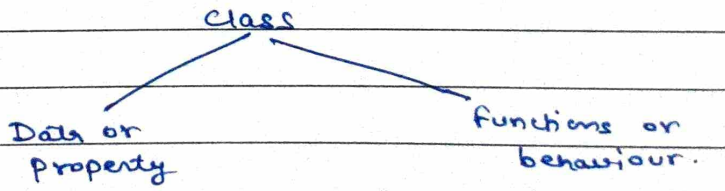
principles of OOP



~~built in class~~
user defined class.

class — A class is a blueprint / template.

object — An object is the instance of the class.



Ex: Class Car
~~Color~~ = "blue"
model = "sport"
def calculate_avg_speed = (km, time)

Object —
~~Car~~ → Wagana R.

pascal case - HelloWorld

def __init__(self) → special function

Constructor is a function which executes when object is called.

~~class ATM~~
~~ATM(pin.~~

class
ATM(int pin, int balance)

~~class ATM~~
~~ATM()~~
~~menu()~~

class diagram

```
Circle
- radius : double = 1.0
- color : String = "red"
----------------------------------
+ Circle()
+ Circle(r: double)
+ Circle(r: double, c: string)
+ getRadius(): double
+ getColor(): String
+ getArea(): double
+ setColor(color: string): void
+ toString(): string  - - - - - "Circle[radius=?, color=?]"
```

```
ATM
- pin : int
- balance : int
----------------------------------
+ ATM()
+ create_pin()
+ change_pin()
+ withdrawl()
+ deposit()
+ exit()
```

Method – function ~~only~~ inside the class

function – function ~~inside~~ outside the class

RANKA
DATE / /
PAGE

Magic Methods | Dunder method :-

magic method → Special → super power

→ __name__

Constructor:- It is a ~~method~~, special method, magic method that ~~reg~~ called when object of that class is created.

Eg.

```
def __init__
```

benefits →

Code written in constructor will not be access by user.

require configuration related code.

Self :-

Keyword that denotes the object in a general sense.

So, If one function want access to another method it is possible using self

Ex:

```
Class Fraction:
    def __init__(self, x, y):
        self.num = x
        self.den = y

    def __str__(self):
        return "Fraction({}/{}).format(self.num, self.den)

    def __add__(self, other):
        new_num = self.num * other.den + self.den * other.num
        new_den = self.den * other.den
        return "{}/{}".format(new_num, new_den)
```

Lec-2 E

RANKA
DATE / /
PAGE

Note:- A class is a combination of magic methods & non magic methods

Lec-2 Encapsulation & Static Keywords

Access attribute through object.

p.gender = 'male'

- Object without reference

```
class Person:
    def __init__(self):
        self.name = 'Kanishk'
        self.gender = 'Male'
```

Person()
p = person() → Here p is reference variable
↳ object

Note:- Reference variable holds the address of object.
It is not necessary to take reference variable for creating object

- Pass by reference.

giving object as a parameter to a function.

```
class Person
    def __init__(self, name, gender):
        self.name = name
        self.gender = gender

def greet(person):
    print("Hello" person.name)
p = Person('nitish', 'male')
greet(p)
```

A function can also take input as parameter (object) also can make object inside function

L_ATM_balance

RANKA
DATE / /
PAGE

In python nothing is private

Encapsulation:

* Instance variable → Special variable whose value is depends upon object & different in objects

· private variable & private attributes

```
class ATM
    def __init__(self):
        self.pin = '' → public
        self.__balance = 0 → private

    def get_balance(self):
        return self.__balance

    def set_balance(self, new_value):
        if type(new_value) == int:
            self.__balance = new_value
```

Encapsulation: A way by which we make our attribute & method private and giving access through getter & setter.

Capsule
Data + Method

Collection of objects:

```
class Person:
    def __init__(self, name, gender):
        self.name = name
        self.gender = gender

P1 = Person('nitish', 'male')
P2 = Person('ankit', 'male')
P3 = Person('anukta', 'female')
L = [P1, P2, P3]
```

Static Variable vs Instance Variable.

```
class ATM
    def __init__(self):
        self.pin = ''
        self.balance = 0.
        self.cid = 0
        self.cid += 1
```

→ Instance Variable

Static variable → class variable

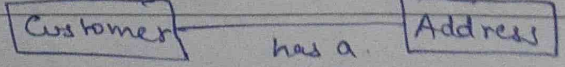
```
class ATM
    __count = 0
    def __init__(self):
        self.pin = ''
        self.balance = 0.
        self.cid = Atm.count
        Atm.count += 1

    @staticmethod
    def get_counter():
        return Atm.__count
```

```
Atm.get_counter()
```

Lec-3 [OOP]

Aggregation [has a relationship]

When one class owns the other class

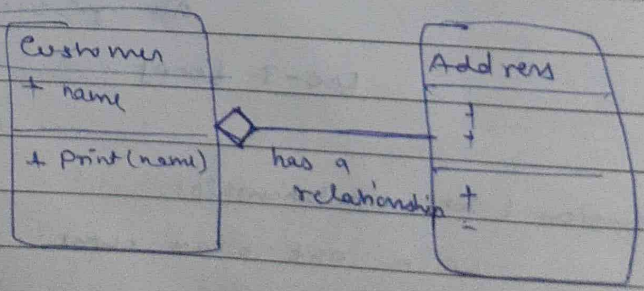


relationship.

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

    def print_address(self):
        print(self.address.city, self.address.state)
        print(self.address.get_city())

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

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

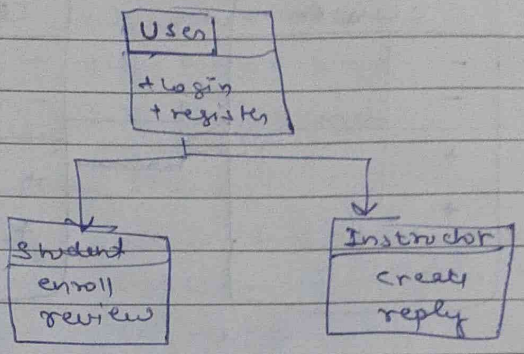
यहाँ Cust class का object बनाते समय Address class का object दिया था।

Aggregation class diagram



Inheritance — child class owns property & attributes of parent class

DRY



Example

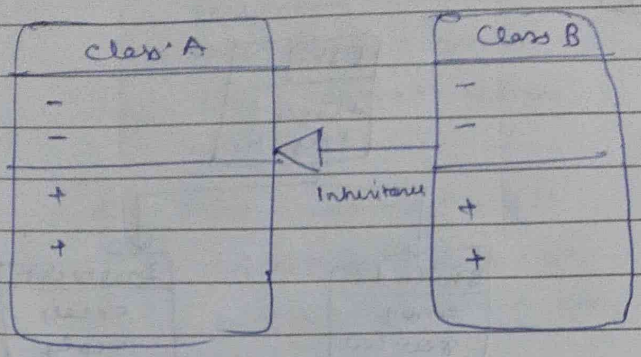
```
class user:
    def __init__(self):
        self.name = 'nitish'
    def login(self):
        print('login')

class student(User):
    def __init__(self):
        self.roll no = 100
    def enroll(self):
        print('enroll into course')

U = User()
S = student()

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

- Method overriding

parent ची constructor call होते आणि child ची constructor नाही

RANKA
DATE / /
PAGE

Inheritance class diagram



What gets inherited.

1) Constructor
2) Non private Attributes
3) Non private methods

Important points.

Method overriding

1) If child has no constructor & method → Parent constructor will call
2) If child has constructor — child constructor will call
3) child can't access private member of parent

Method Overriding:

If their is same functionality in child & parent then preference will be given to child functionality

Super keyword:

Super is a way to access parent method / functionality when they have same functionality to break preference

Note: Super keyword can be call inside child 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 __init__(self, price, brand, camera, os, ram):
        super().__init__(price, brand, camera)
        self.os = os
        self.ram = ram
        print("Inside Smartphone constructor")
    def buy(self):
        super().buy()
s = Smartphone(20000, "Samsung", 12, "Android", 2)
print(s.os)
print(s.brand)
```

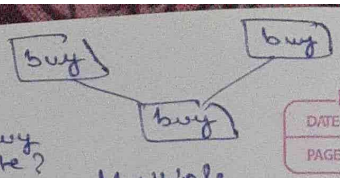
Inheritance

- class can inherit from another class.
- Improves code reusability
- Constructor,
- parent has no access to child,
- private property of parent not access to child.
- child class can override attribute or method of parent

super() is a built in function which is used to invoke the parent class method & constructor

Types of Inheritance

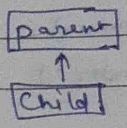
- Single
- Multilevel
- Hierchial
- Multiple - (Diamond prob)
- Hybrid



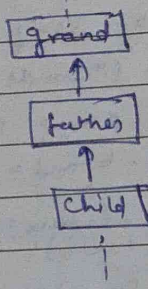
RANKA
DATE / /
PAGE

Which buy execute?

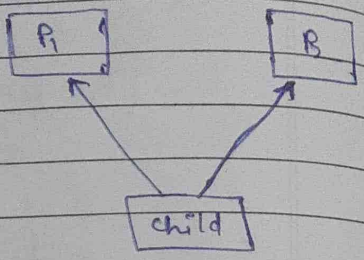
Single Inheritance



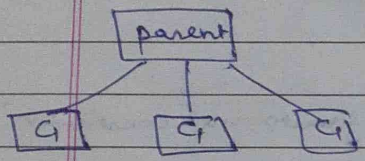
Multilevel



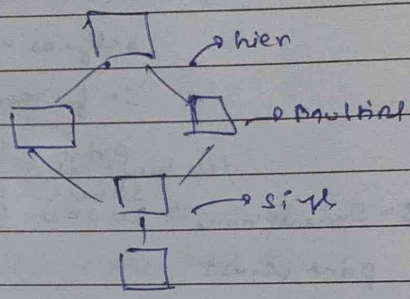
Multiple
(Not in Java) ~~Hierarchical~~



Hierarchical (श्रेणीबद्ध)



Hybrid



Not

Multilevel

```
class Product:
    pass

class Phone(Product):
    pass

class Smartphone(Phone):
    pass
```

Multiple

```
class Phone
    buy()

class Product
    buy()

class Smartphone(Phone, Product)
    pass
```

यहाँ किसका buy execute होगा?
जिसका नाम पहले लिखा है

(Method Resolution Order)

RANKA
DATE / /
PAGE

Polymorphism :- having multiple forms.

If a function behave differently depending on situation

1) Method Overriding
2) Method Overloading - different behaviour in terms of input
3) Operator Overloading -

Not work in Python

Method Overloading :- The code is more clean

```
class shape:
    def area (self, radius):
        return 3.14 * radius * radius

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

Instead of this

```
class shape:                          # using default keyword.
    def area (self, a, b=20):
        if b==0
            return 3 * r * r
        else:
            a*b.
```

Operator Overloading :- [magic method]

'hello' + 'world' - hello world

4+5 - 9

Child class ફરજી કામ કરાવે)

Abstraction: hidden. (છુપા કરવા)

Higher class can make it compulsory to lower class to do some task.
(કંઈક કામ કરાવે)

```python
from abc import ABC, abstractmethod
class BankApp(ABC):
    def database(self):
        print('connected to database')

    @abstractmethod
    def security(self):
        pass

class MobileApp(BankApp):
    def mobile_login(self):
        print('login into mobile')

mob = MobileApp()
```

Note: Abstract class cannot have object