Agenda

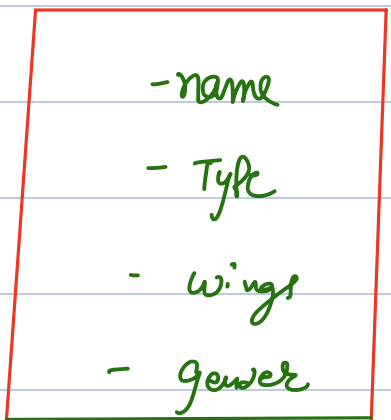
SOLID: Guidelines & fundamentals

1. Extensibility
2. modularity
3. maintainability
4. readability

Today
- S : Single Responsibility
- O : open close Principle
- L : Liskov's Substitution Prin.

Tomorrow
- I : Interface Segregation
- D : Dependency inversion

BIRD VI



-eat(), -fly(), -make sound()

b = Bird()

b.name ___

b.Type = Peacock

b.wings = 2

b2 = Bird()

b.name ___

b.Type = Sparrow

b.wings = 2

b1. make_sound() # Peacock sound

b2. make_sound() # Sparrow sound

Bird:

def make_sound( ):

if Type = = Peacock:

play_Peacock mp3;

if Type = = Sparrow:

play Sparrow mp3

1. Not_readable
2. difficulty Testing
3. NOT extensible
4. NOT following 'S' of Solid

. . .

S: Single Responsibility Principle

Every code Unit (class, method, package)
Should have only 1 responsibility

only 1 reason to change.

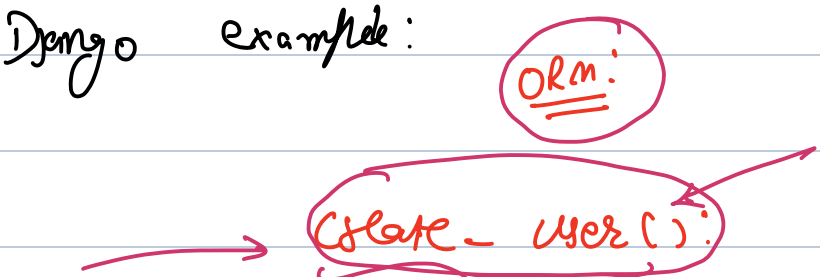
ways To find if S is broken

Too many if conditions

```
command_if_do(command):
    if dang bees:
        Restart

    if short_bees:
        Shut down
```

Django example:



S.R.P ✗ | User.Object.Save()
send_email()

O.C.P : Open - Close Principle

your code should

Open for extension & closed for modification

add new features

never change the existing code

OCP ✗

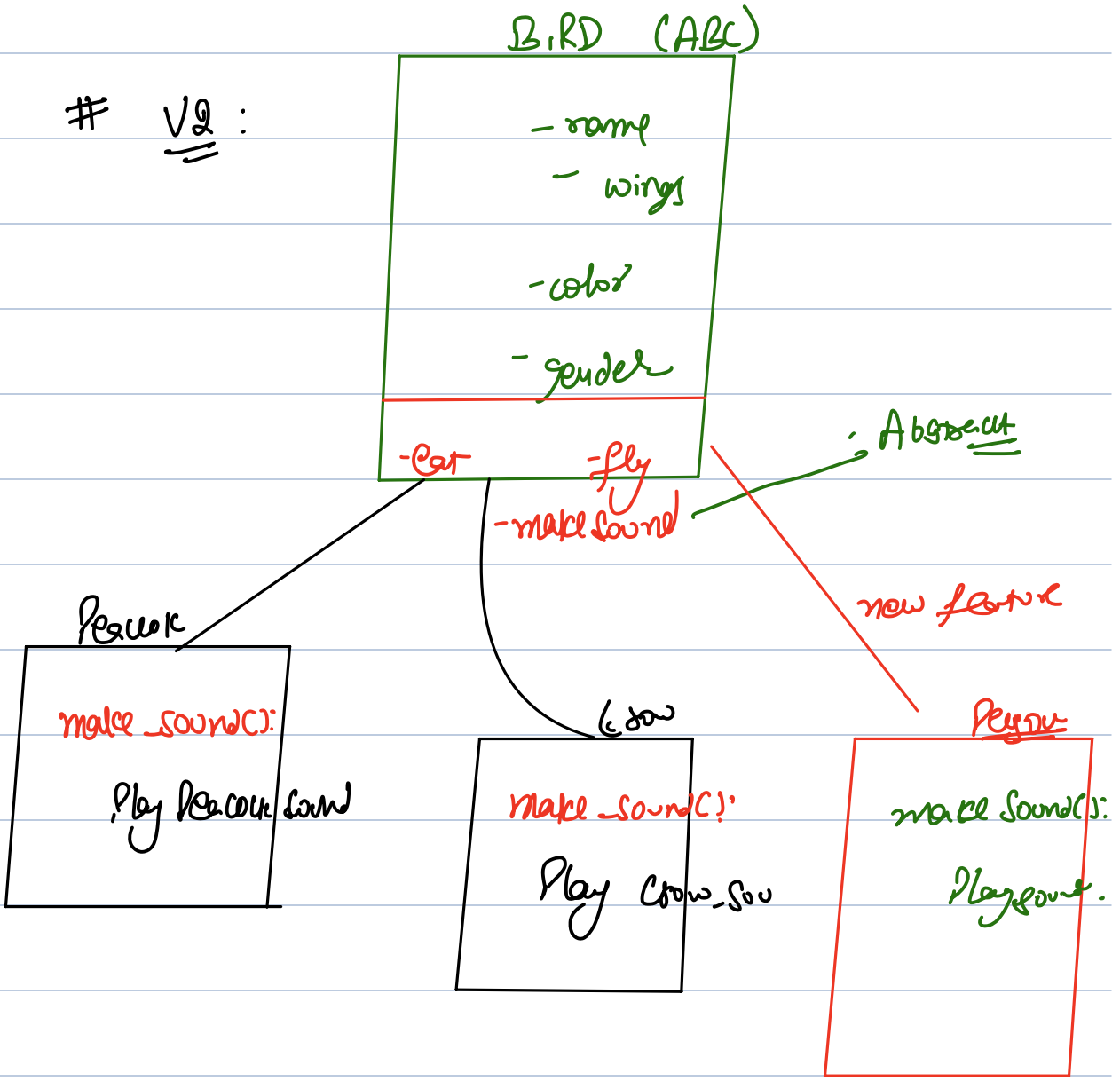
def make_sound():
    if type == 'P':
        ...

we are changing existing code

To add a new feature

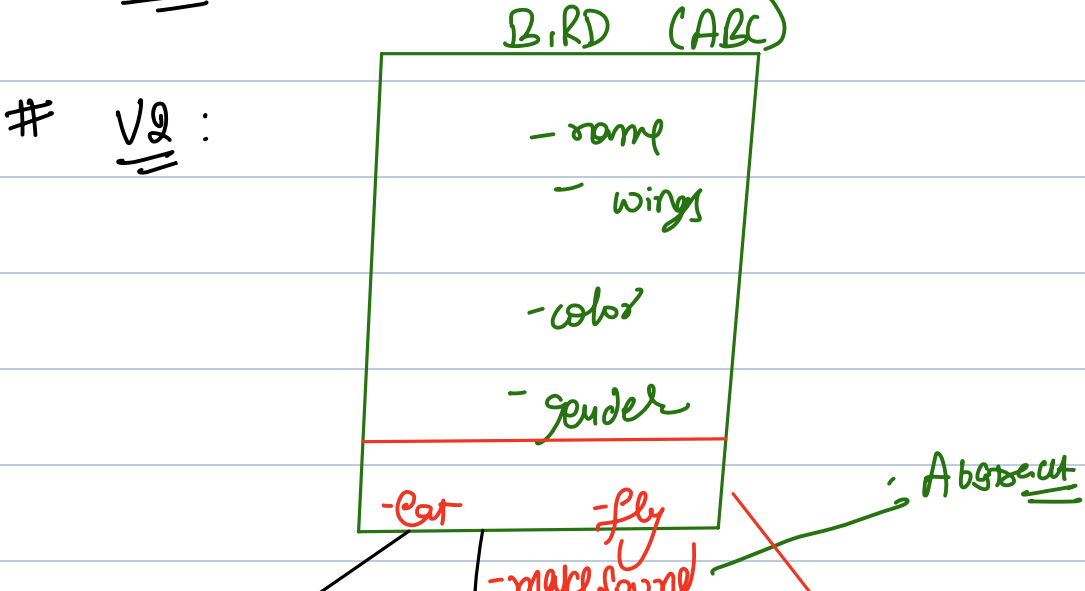
if Type == 'Owl':
    ...

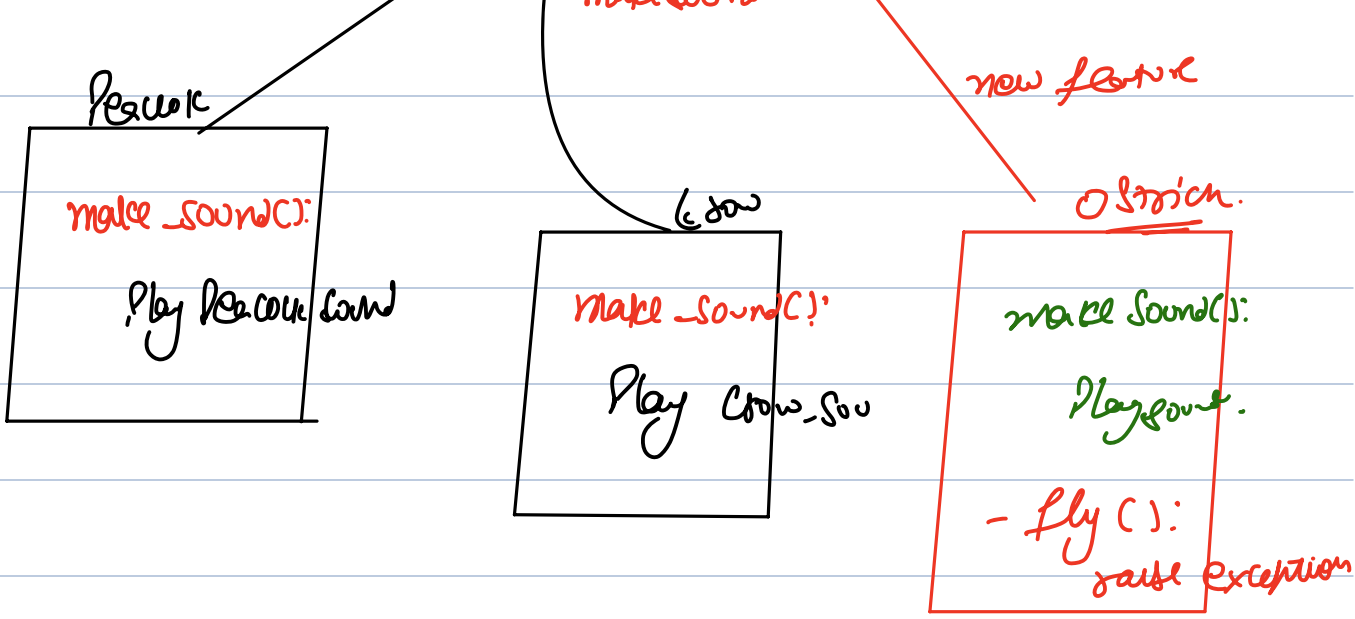
V2 :


BIRD (ABC)
- name
- wings
- color
- gender
-eat
-fly
-make sound
: Abstract
now flow
Peacock
make_sound():
Play Peacock sound
Cow
make_sound():
Play Cow_Sou
Penguin
make sound():
Play sound.

LSP :

V2 :

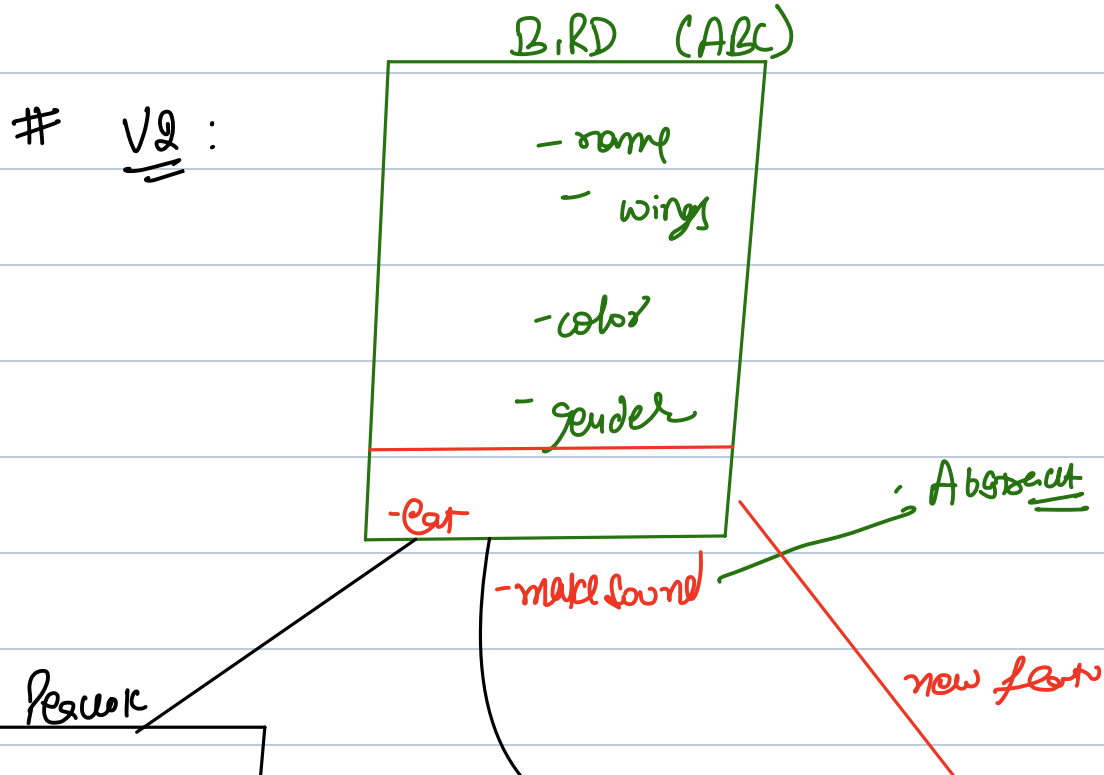

BIRD (ABC)
- name
- wings
- color
- gender
-eat
-fly
-make sound
: Abstract

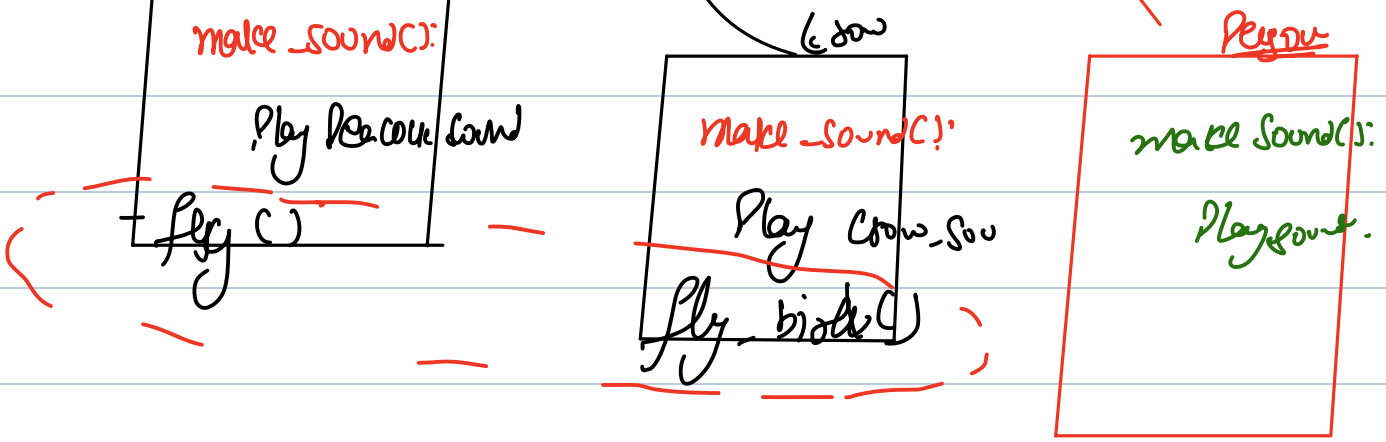



```
def Race_of_birds(birds):
    for B in birds:
        B.fly(). # fails
```

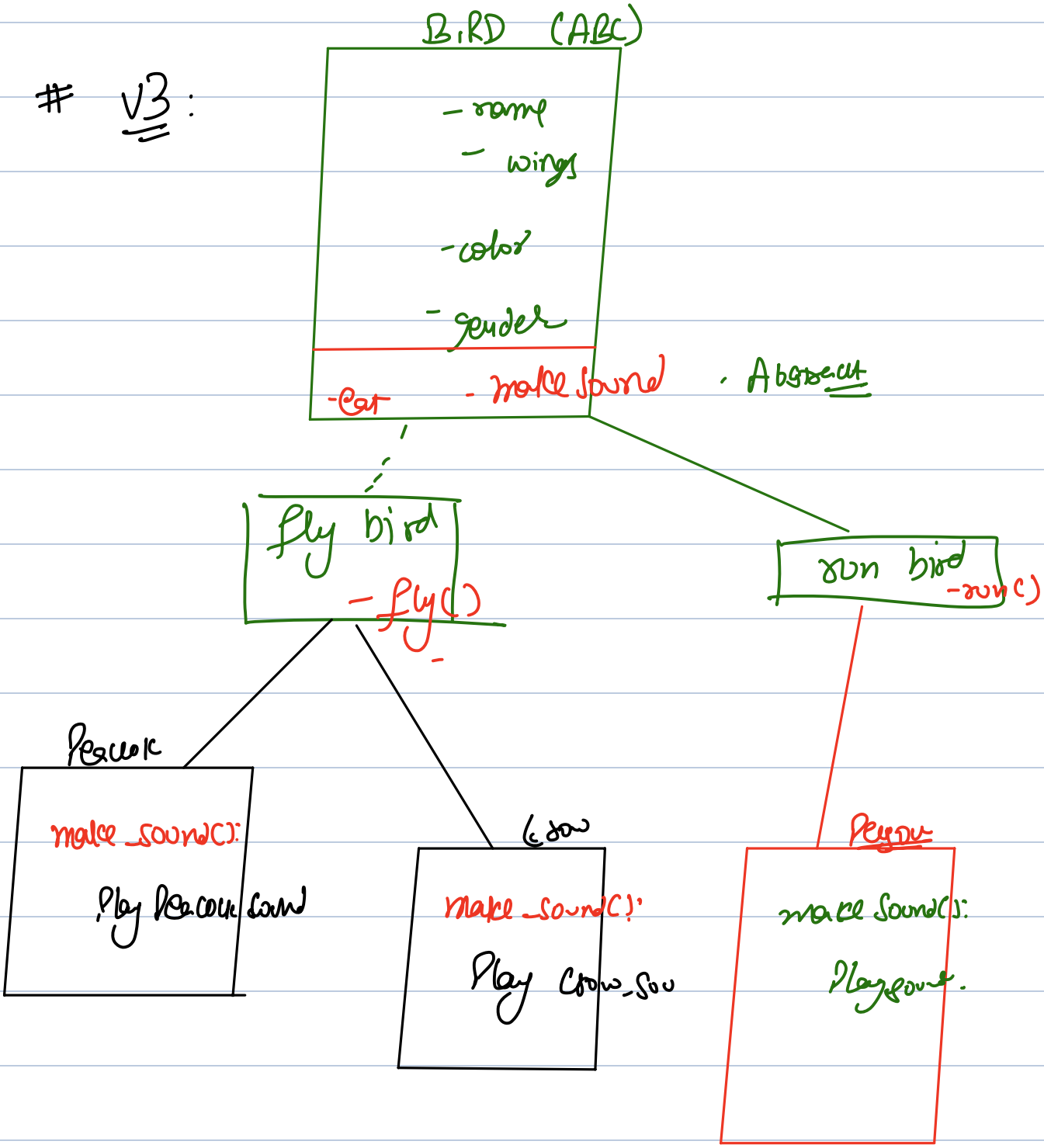
```
race of birds([Peacock, ostrich])
```

V2 :




make_sound():
Play Peacock sound
fly()
Crow
make_sound():
Play Crow_Sou
fly_bird()
Peacock
make sound():
Playsound.

V3:


BIRD (ABC)
- name
- wings
- color
- gender
- eat
- make sound
. Abstract
fly bird
- fly()
run bird
- run()
Peacock
make_sound():
Play Peacock sound
Crow
make_sound():
Play Crow_Sou
Peacock
make sound():
Playsound.

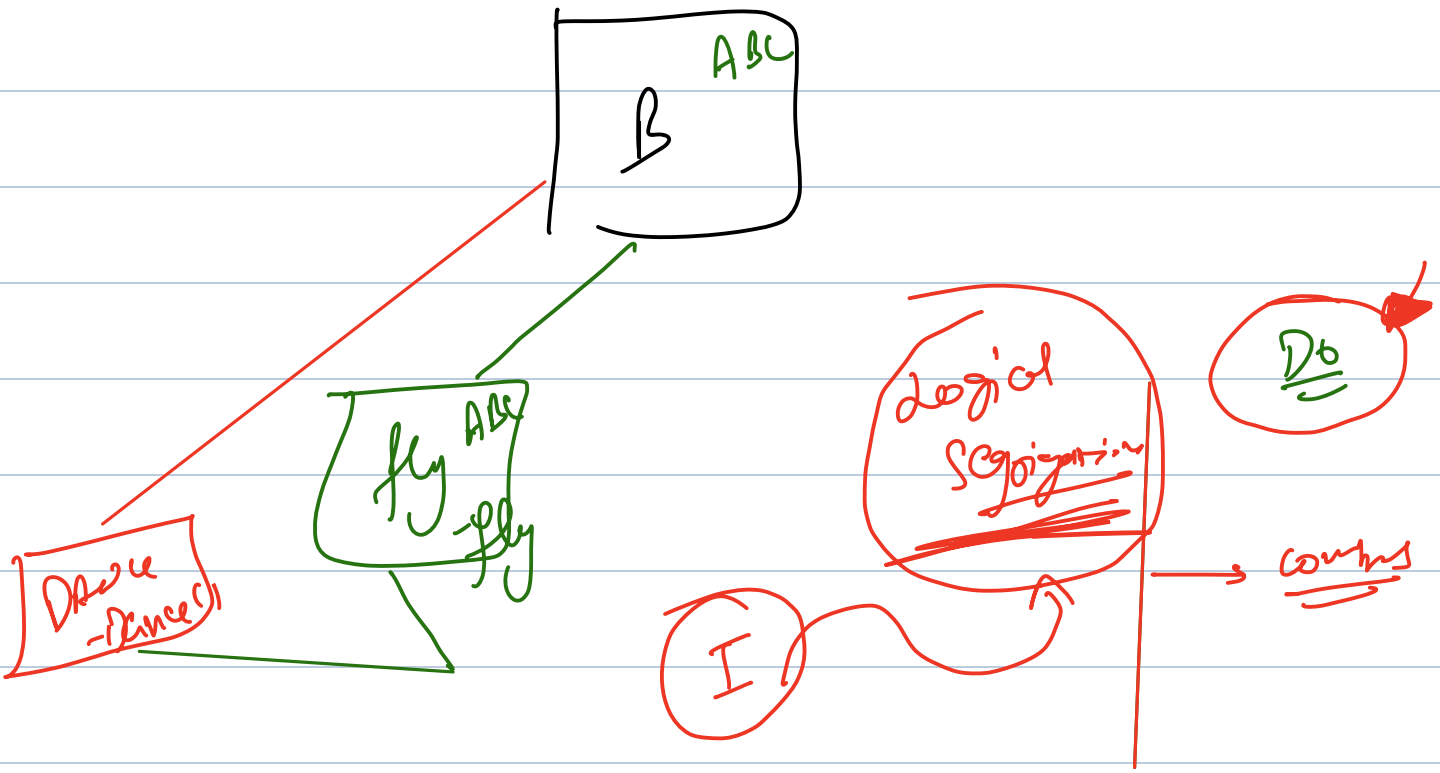
Never ever explore the method which is not your feature.

ensure Subclass can be replaced from Parent Class without behaviour of Program being altered

A = ~~Animal~~ Dog () = bark, ~~sleep~~, eat

A.bark()
A.sleep
A.eat()

Dog() — bark, sleep, eat, give birth


B
ABC
fly
ABC
fly
Dance
Dance
Logical
Segregation
Do
I
comms