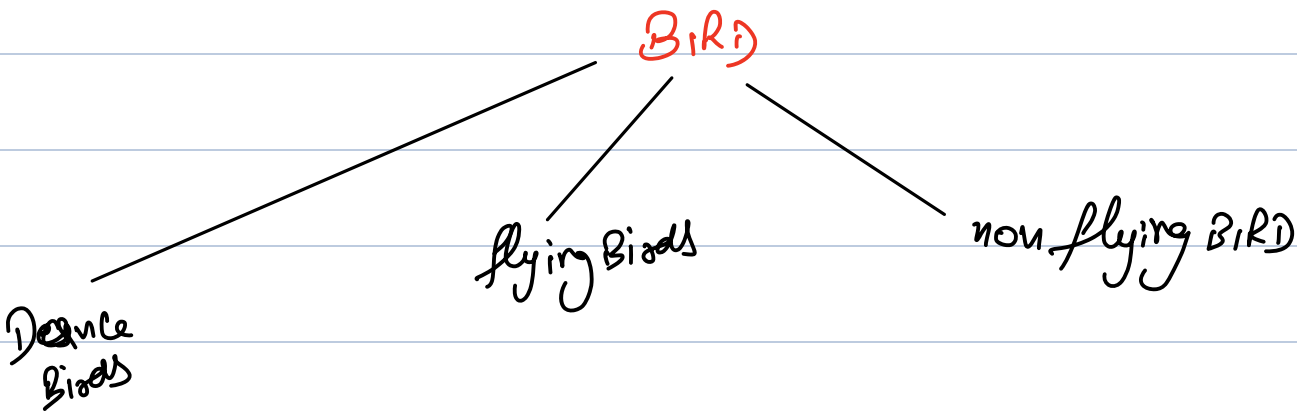
Agenda :

- SOLID
- Design Patterns
- Singleton

I : Interface Segregation



Single Responsibility Principle.

Keep interface as light as possible.

Dabase interface

- read
- update

- Create

- Delete

DIP : Dependency inversion Principle.

No & Concrete classes should depend on each other, they should talk via Abstract classes.

```
Class switch():
    light : Electrical = light()        Refactor too()

    def on()
        - light.Turn on()

    def off()
        - light.Turn off()
```

Abc Electrical Alliance()

Dy Turn on()

Dy Turn off()

Design Patterns:

Something repeating

Software

Why design Pattern:

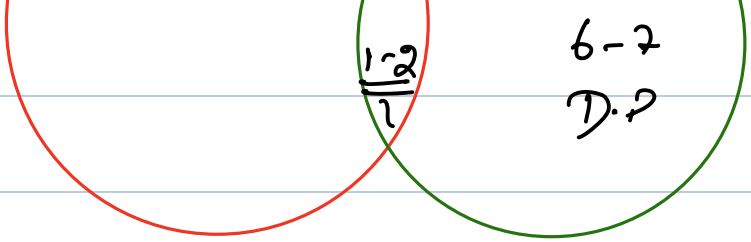
1. Common Vocab
2. Save a lot of time.

≃ 23 D.P

≃ 10-11 D.P

{ exception_dev }

3-4 D.P ⊆ Interview



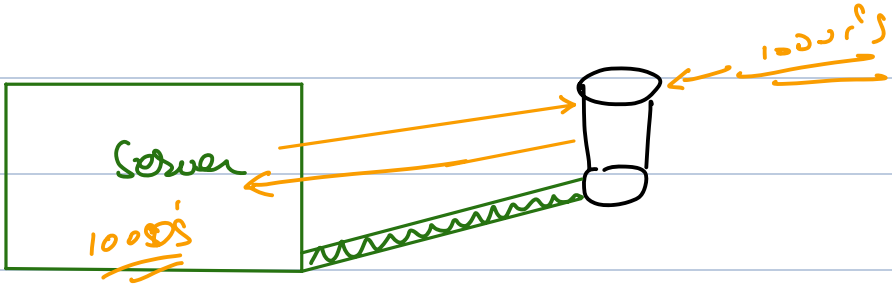
Type of D.P?

1. Creational — focus on Obj creation
2. Structural — How to Structure classes
3. Behavioural. — How to behave in a class.

Creational:

1. Singleton
2. Builder
3. Factory
4. Prototype
5. Registry

Singleton D.P:



Class Db connection :

def __init__()

return old object

d1 = Db connection()

d2 = Db connection()

Class Db connection :

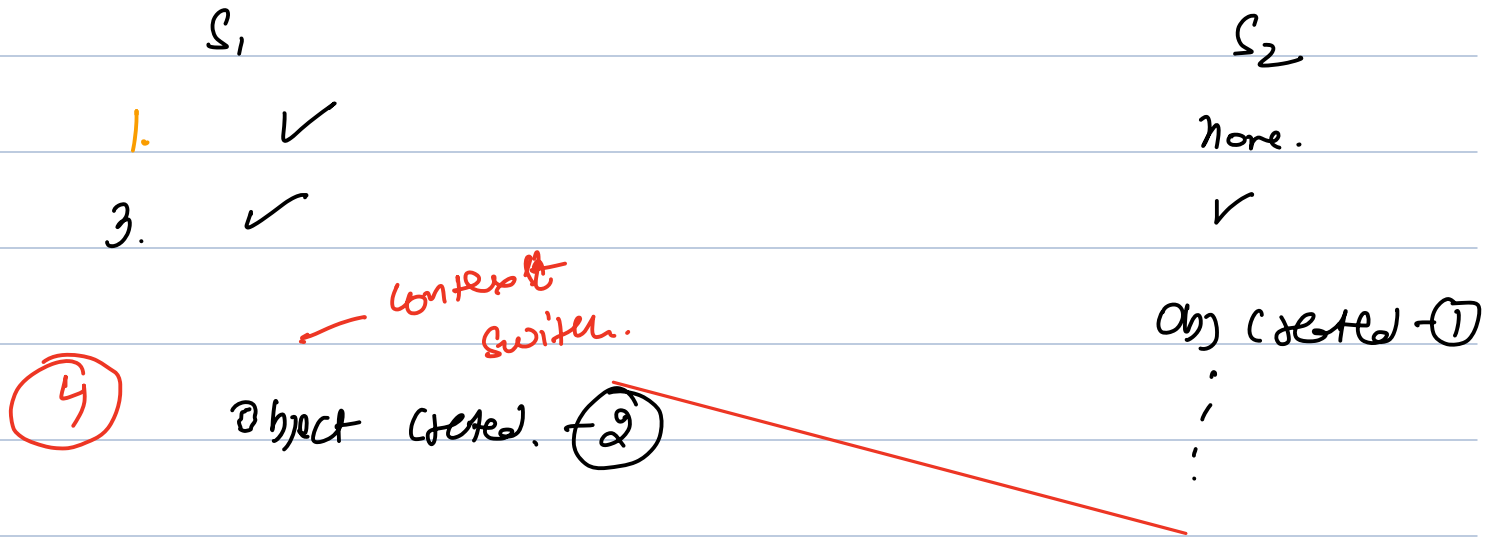
1 __instance = None

2 def __new__(cls):

3 if __instance is None:

4 __instance = super().__new__(cls)

5 return __instance;



How To Solve :

```
Class  Db connection :
1       __ instance = None
2       def __new__ (cls):
        with lock:
3           if __instance is None:
4               __instance = super().__new__
                ()
5       return __instance;
```

Class Db connection.

```
1   __instance = None
2   def __new__(cls):
2'' if cls.__instance is None:
3''     with lock:
3           if cls.__instance is None:
4               cls.__instance = super().__new__(cls)
5       return cls.__instance
```

S_1	S_2
1. None	Obj ~~None~~
2 ✓	2 ✓
2'' ~~✓~~	2'' Tom.
— context switch	3'' ✓
3'' ✓	3 ✓ Object created
3 false	4 ✓
5 return	5 ✓