Agenda:

→ L
→ I } → of SOLID
→ D

→ Dependency Injection

Example

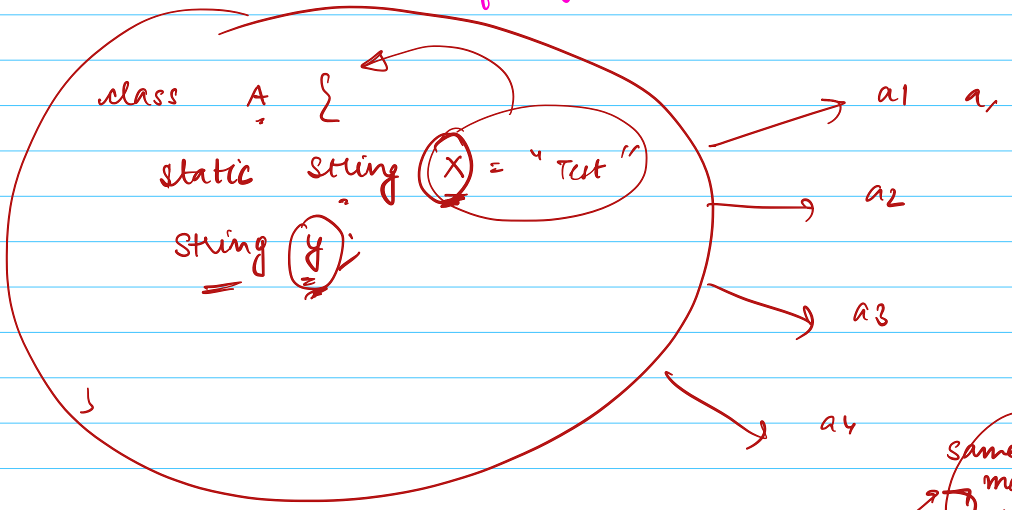
Utils → (all static methods)

inp → output

static → variables, methods

↳ classes → Builder pattern

Associated with class instead of objects.



```
class A {
    static String x = "Test"
    String y;
}
```

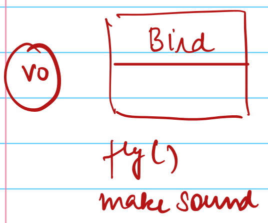
a1.x = "Test2)

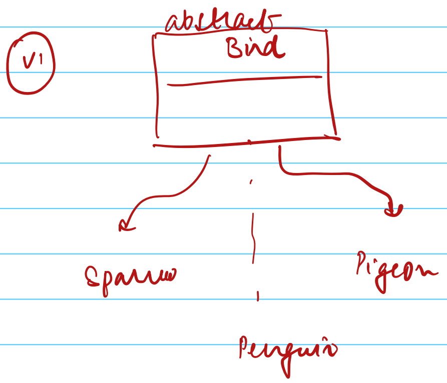
a2.x also changed

a1.x
a2.x → same memory location

a1.y != a2.y

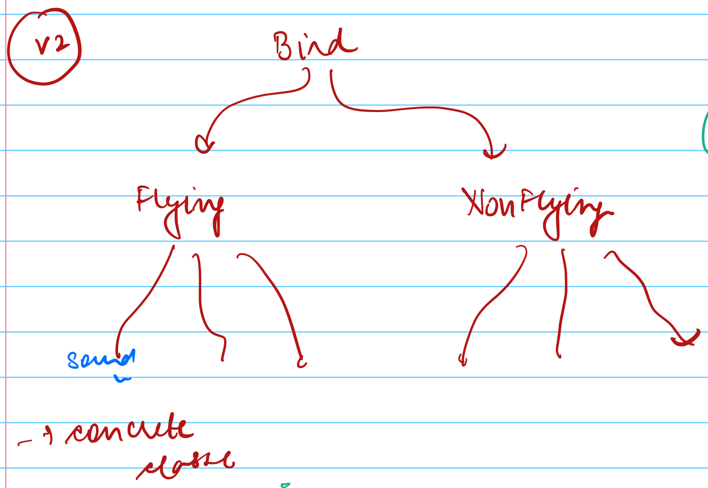
RECAP





SRP

OCP ✗



① Class Explosion
↳ 2^n

② We were not able to get a common type for a behaviour

{Example for birds that make sound ?}

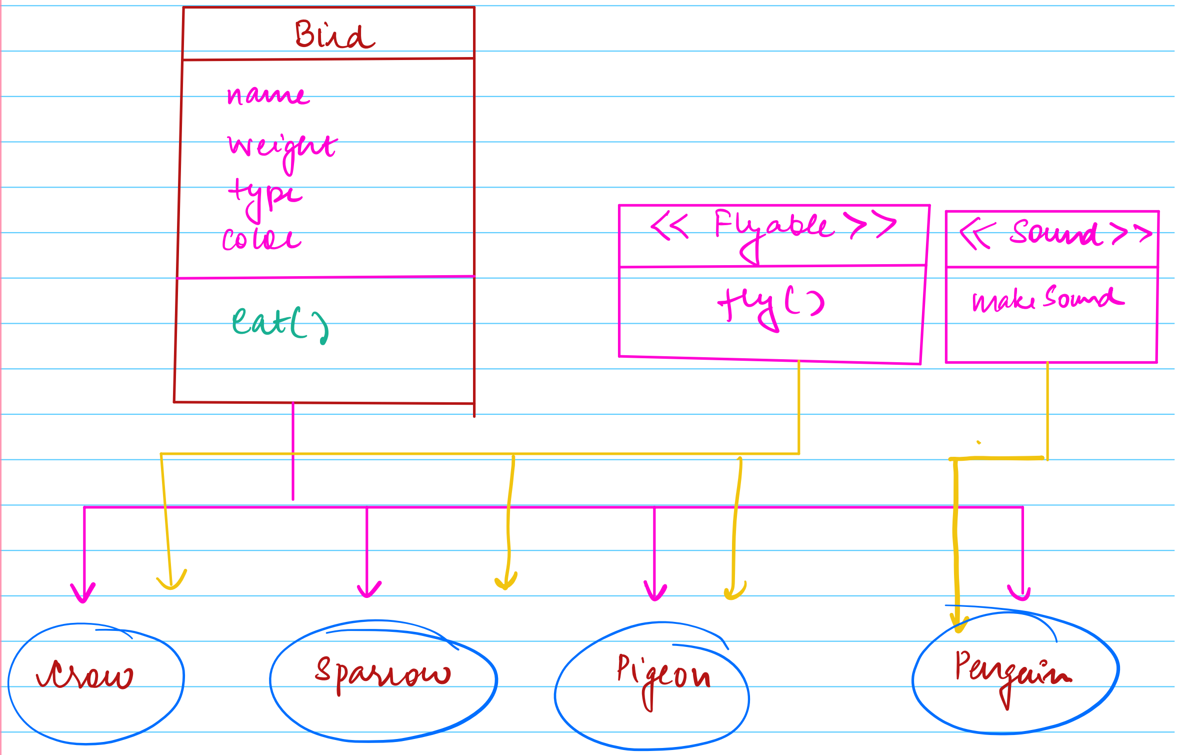
Behaviours → Abstract classes

↓

Interfaces

Some birds have a behaviour and some don't ⇒ {Implement those behaviours via interface.}

abstract class
→ Every bird has some attributes?



```
class Pigeon extends Bird
        implements Flyable, Sound {

    fly() {
    }
    make Sound() {

    }
}
```

List < Flyable > birds . — -

⇒ birds.fly()

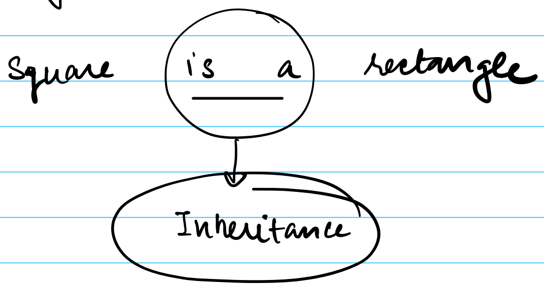
Liskov's Substitution Principle [LSP]

We should be able to replace references of parent class by its subclass without breaking the code

OR

No subclass should provide special meaning to the methods of superclass

Rectangle   Square



```
class Rectangle {

    int height;
    int width;

    void setWidth().
    void setHeight()

    int getHeight()
    int getWidth()
    int getArea()

}
```

```
class Square extends.
        Rectangle {

    @ setWidth
         and
      setHeight →

    getArea
      ✗ . -

}
```

We ended up giving a special meaning to the method of super class.

~~Rectangle~~ rectangle = new Rectangle(4,6)

Square

.get → Break things or give unexpected results

~~Bird~~ b =

Penguin

b. fly

not following LSP

<< Runnable >>

run( )

Human implement Runnable

run( ) }

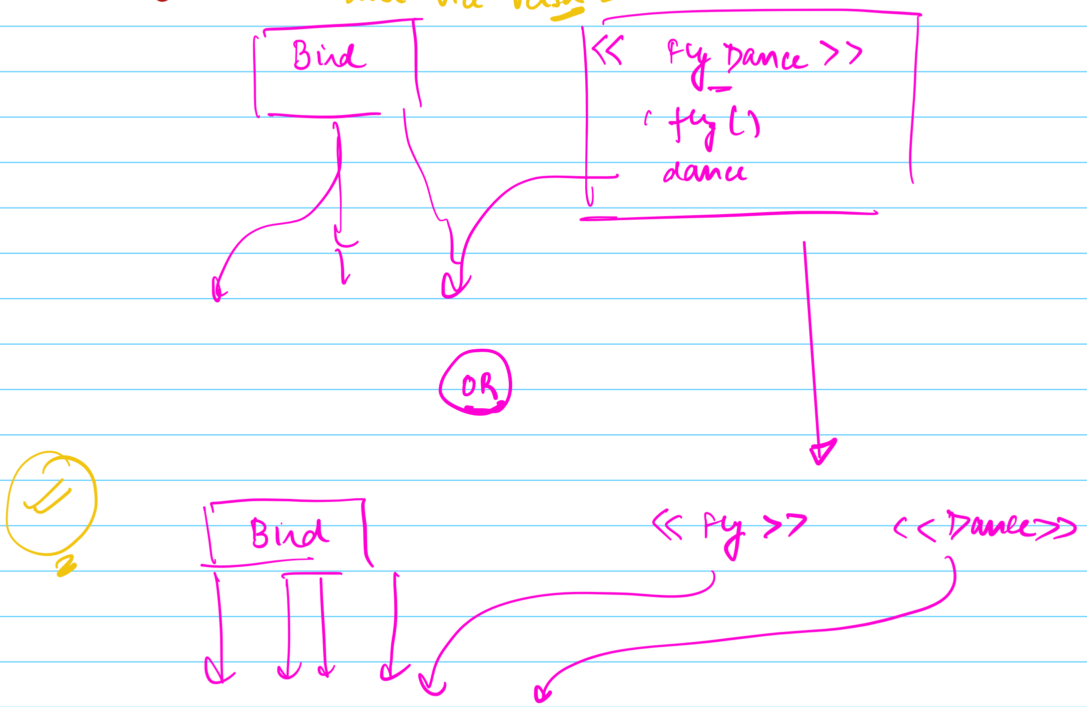
Usain bolt

}

INTERFACE SEGREGATION PRINCIPLE (ISP)

PM:

1. Some birds can fly
2. Some birds can dance
3. Whoever flies, dances. and vice versa



→ Interfaces should be as light as possible

→ As less behaviours as possible

(*) Functional Interfaces

↓

Ideal → 1 method in an interface.

↳ working with lambda

We should only keep those behaviours together in an interface which are very well related / bound to each other

File Reader

open()
close()
read()

SRP on interfaces

DEPENDENCY INVERSION PRINCIPLE

Case Study

Phonpe

↓

(YesBank)

class Balance Checker {

Banking Service ← ~~YesBank Client~~ yb = new YesBankClient();

checkBalance ( )

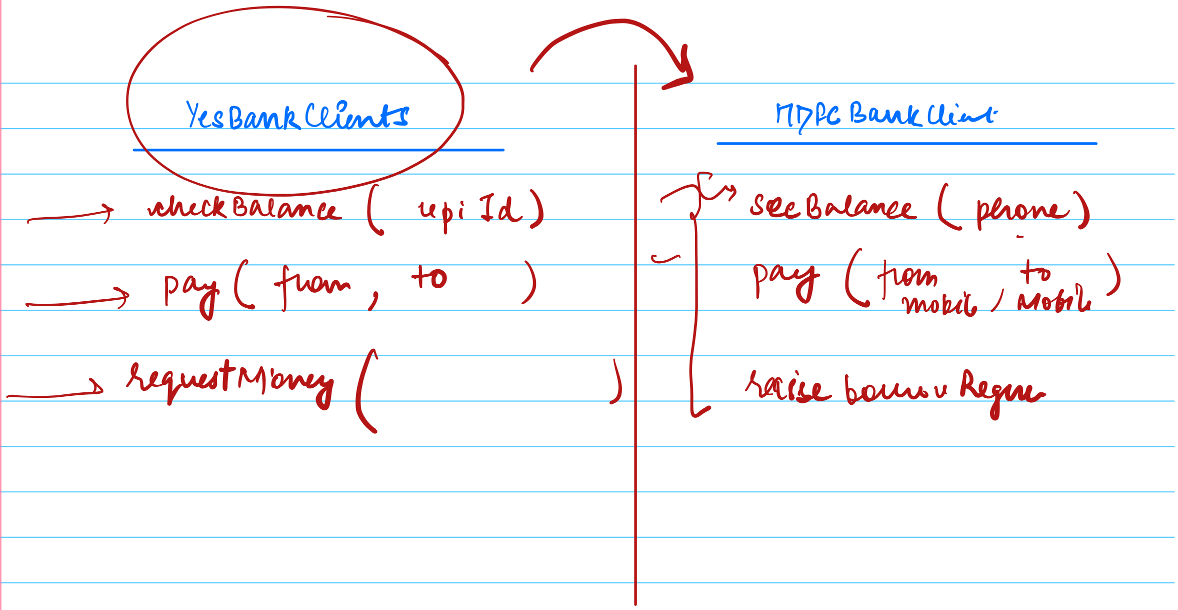
yesBank Client . checkBalance (upi)

establish Connection

authorize()

}

YesBank → ICICIBank



YesBankClients

- checkBalance (upiId)
- pay(from , to)
- requestMoney ( )

HDFC Bank Client

- seeBalance (phone)
- pay (from mobile / to mobile)
- raise borrow Reque

Coded to a Concrete Class

Balance Checker ⟶ Yes BankClient

> No 2 concrete classes should be directly depending on each other
>
> They should depend on each other via an interface

interface Banking Service {

checkBalance)

pay( )

request Money( )

}

YesBank Handler implements BankingService

HDFC Bank Handler imple -

↓

class HDFC Bank Handler implements Bank

— check Balance (—)

hdfBankClient . seeBalance (phone number)

Sparrow
Pigeon } → Low Flying Behavior

crow,
Eagle } → High Flying Behavior

Pelican } → Water flying Behavior

All classes
implement fly method

class Sparrow extends Bird
implements Flyable

Flying Behavior fb = new Low Flying Behavior ( )

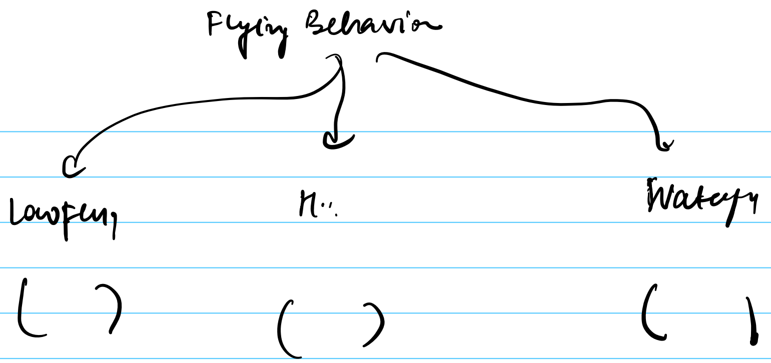
Interface
Ab

void Fly ( ) {

fb . fly ( )

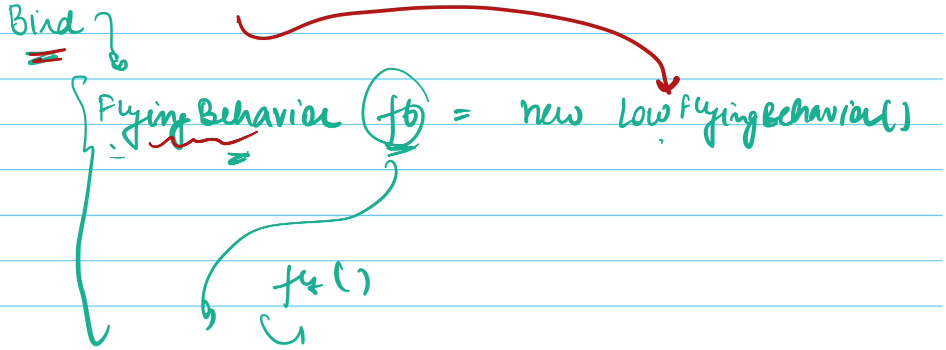
Fly

}



STRATEGY DESIGN PATTERN

DEPENDENCY INJECTION
↓
NOT SOLID principle



```
FlyingBehavior fb = new LowFlyingBehavior()
```

Our responsibility to invoke the FlyingBehaviour instance;

↓ Dependency Injection

Delegate responsibility to client

FB

```
Sparrow
static
    Flying Behavior fb
        = new ___

    fly ( ) {

        fb. makeFly()

    }
```

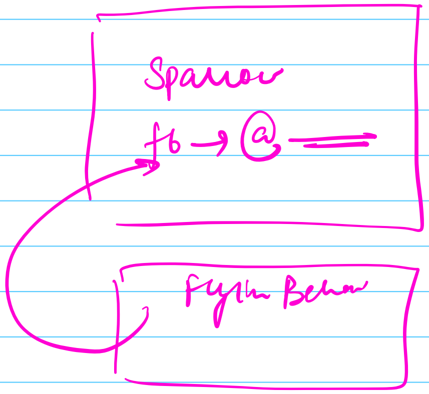
DI

```
Sparrow {

    Flying Behavior fb

    Sparrow( Flying Behavior fb)
        this.fb = fb;

    fly( )
        fb.makefly;
    }
}
```

Spring Boot | Django

Autowired

@ component



Dependency Injection helps achieve Dependency Inversion