Today's agenda

↳ SOLID Principle

* good code → should not have bugs
  - ↳ Maintainable
  - ↳ Reusable → modular
  - ↳ Easy to test
  - ↳ should be configurable
  - ⋮

↳ SOLID design Principles
  - ↳ rules / expectations / guideline

S → Single responsibility Principle
O → open close Principle
L → Liskov's substitution
I → Interface Segregation
D → Dependency inversion

→ LLD Design
  - ↳ subjective concept

→ Design a Bird repo
  - ↳ Design a system which will store info about all the birds on the Planet.

80

```
Class Bird {
    Color;
    weight;
    name;
    age;
    breed;

    CollectFood();
    buildNest();
    makeSound();
    fly();
}
```

```
Bird b1 = new Bird();
b1.Color = black;
b1.name = "crow";
b1.wight = 200;
```

```
Bird b2 = new Bird();
b2.Color = White;
b2.name = "Pigeon";
b2.wight = 300;
```

```
fly() {
    ≡
}
```

b1.fly()

b2.fly()

DRY

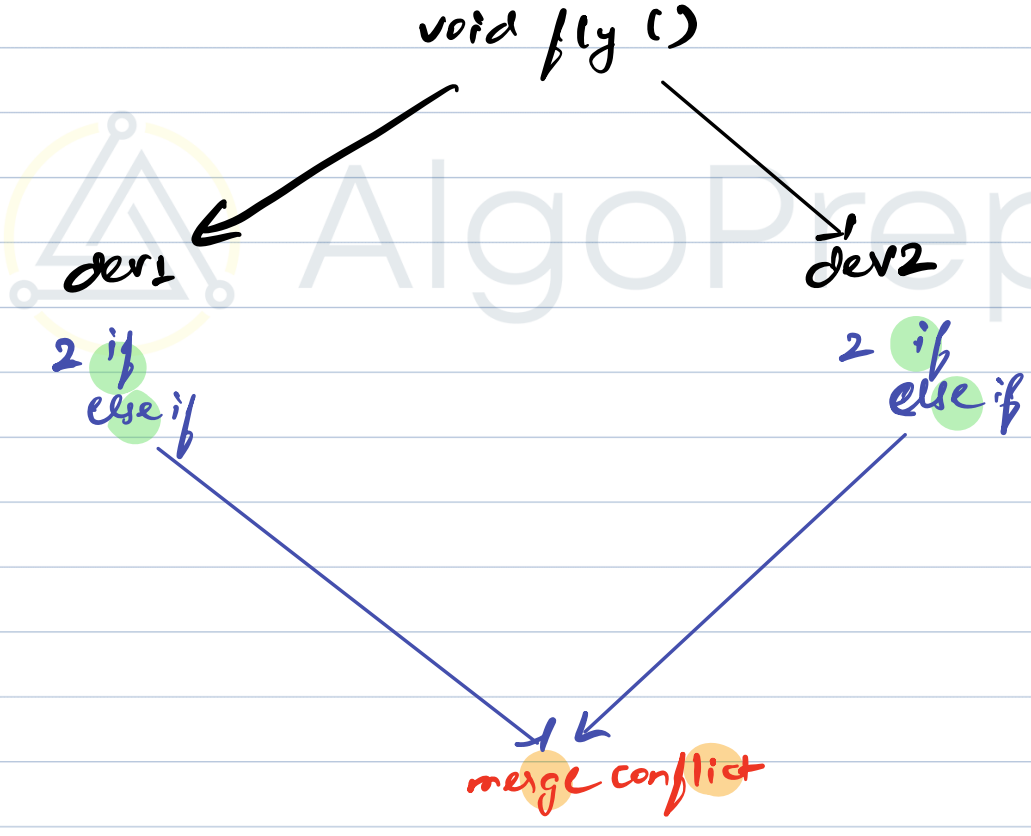
```
void fly() {
    if (name == crow) {
        ≡
    }
    else if (name == Pigeon) {
        ...
    }
    else if (      ) {

    }
    ...
    else if (cond) {
        ≡
    }
    else {
        ...
    }
}
```

→ How each type of bird should fly.

Parrot

Problems:

- (i) Difficult to understand.
- (ii) Extensibility difficult.
- (iii) Difficult to test.
- (iv) Code duplication
- (v) merge conflict



(vi) violates SRP (Single responsibility Principle)

(vii) violates OCP (open close Principle)

→ Single Responsibility Principle

Is every code (method / class / package) must have exactly 1 defined responsibility.

↓

why someone should edit the code of that class / method.

mvc

controllers / component

Is user controller

Is group controller

services

Is user services

models

Is Movie

Is seat

Identify violation of SRP

① Methods with multiple if-else.

exception

```
check leapyear (int y) {
    if (y % 4 == 0)
        leap year
    else (y % 100 == 0)
        leap year
    else if (y % 4 == 0 && y % 100 == 0)
        leap years.
    else
        non leap year
}
```

(11) Monster method → large

↳ method in which the code is doing more than what its meant to be doing.

```
Saveto database ( . . . ){
    { Calculate something . . .
      . . .
      .
    { Create db connection
    [ db. execute (query);
}
```

Saveto database (. . .){

calculate ();

createconnection ();

[db. execute (data);

}

{ Private in calculate (){

= =

Integration solving

}

Private void createconnection()

reusable

}

(3) Common / utils folder

↳ garbage place of your codebase.

→ java.util

↳ fetch date

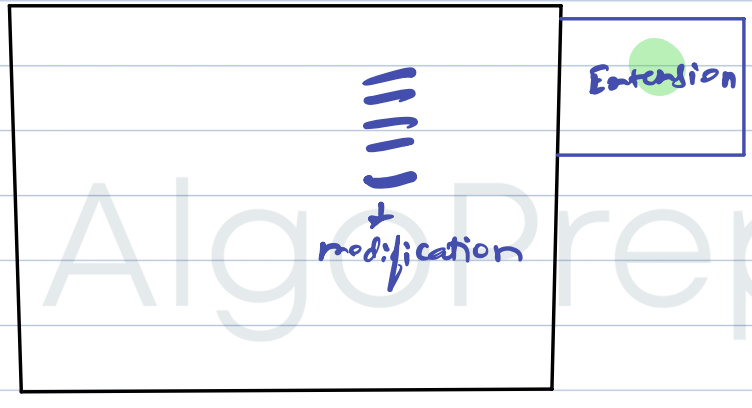
↳ create map

→ Open Close Principle

↳ Codebase should be open for extension but close for modification.

↳ adding feature should require minimal changes to the existing code.

↓
easy to add new features



Break till 9:30 Pm

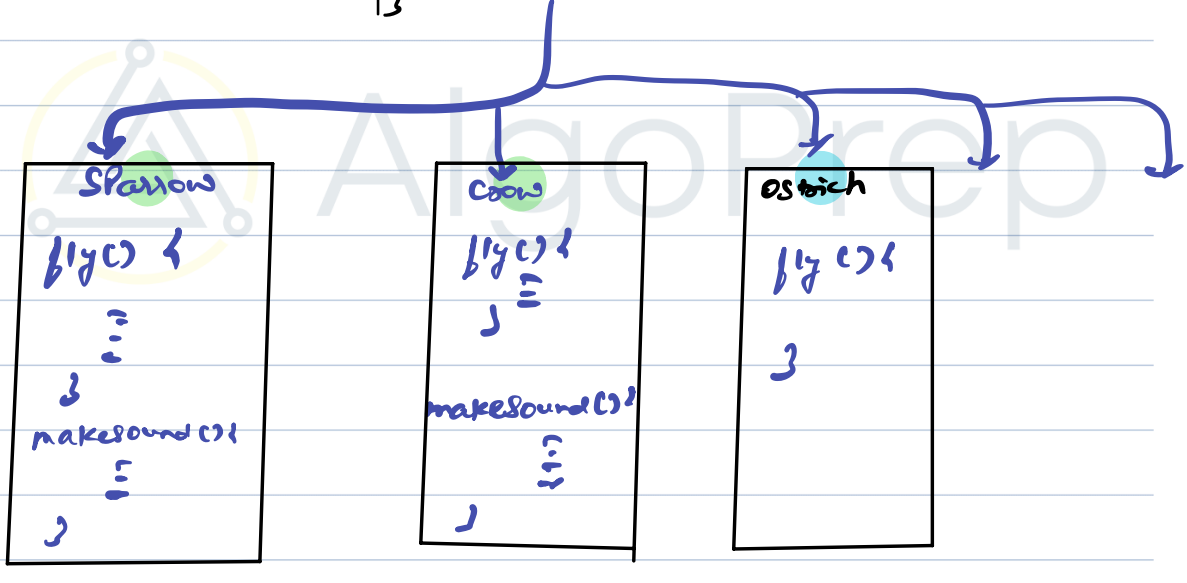
SL

abstract class Bird {

color;
weight;
name;
age;
breed;

abstract collectFood();
abstract buildNest();
abstract makeSound();
abstract fly();

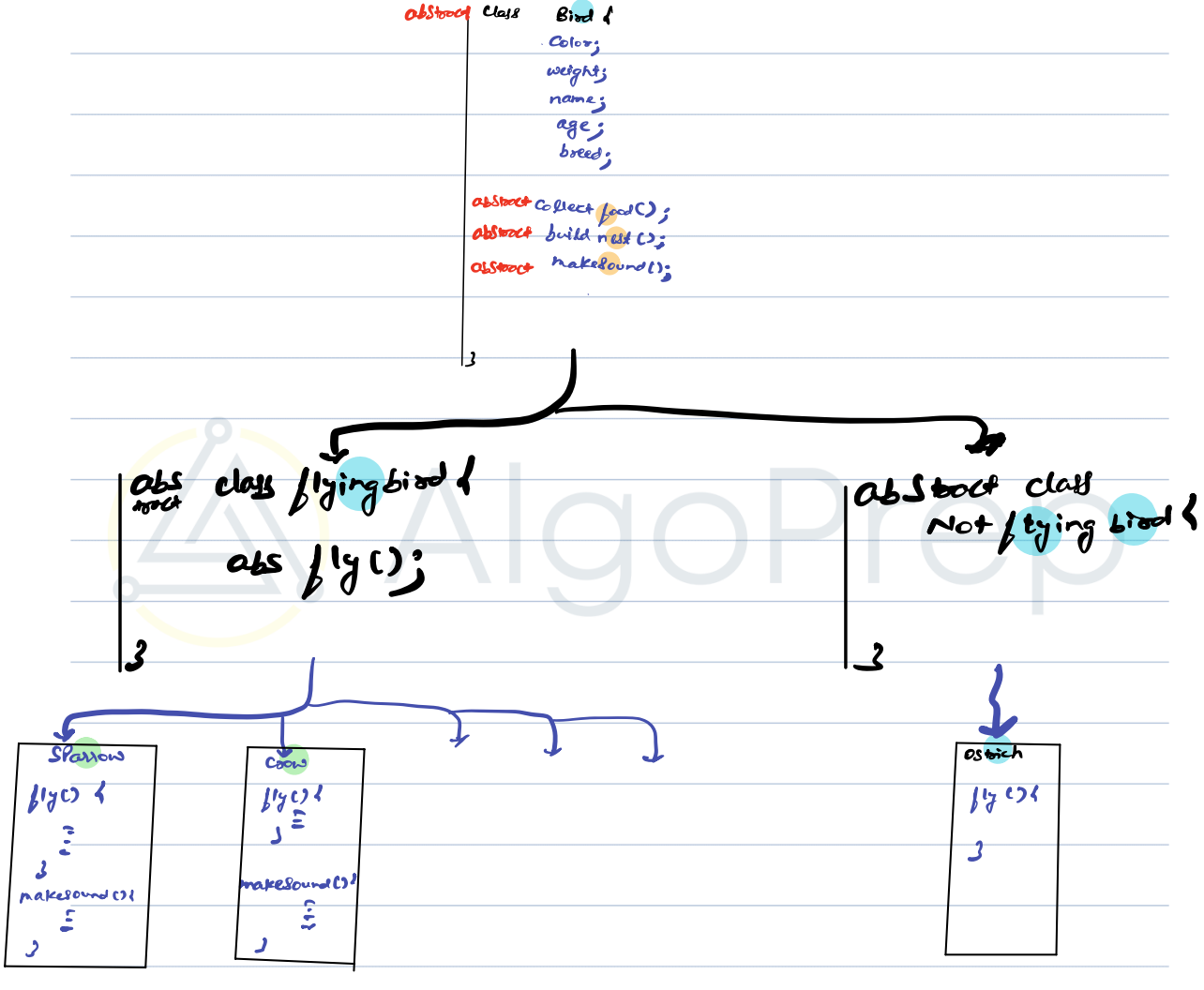
}



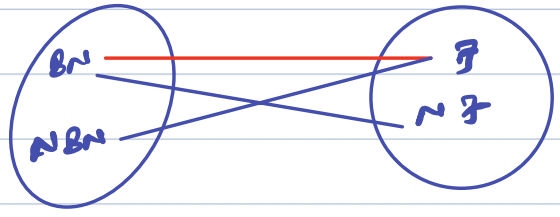
SRP

OCP

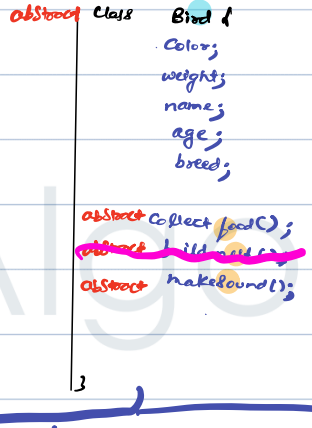
* Penguin, Ostrich
→ birds which can't fly.

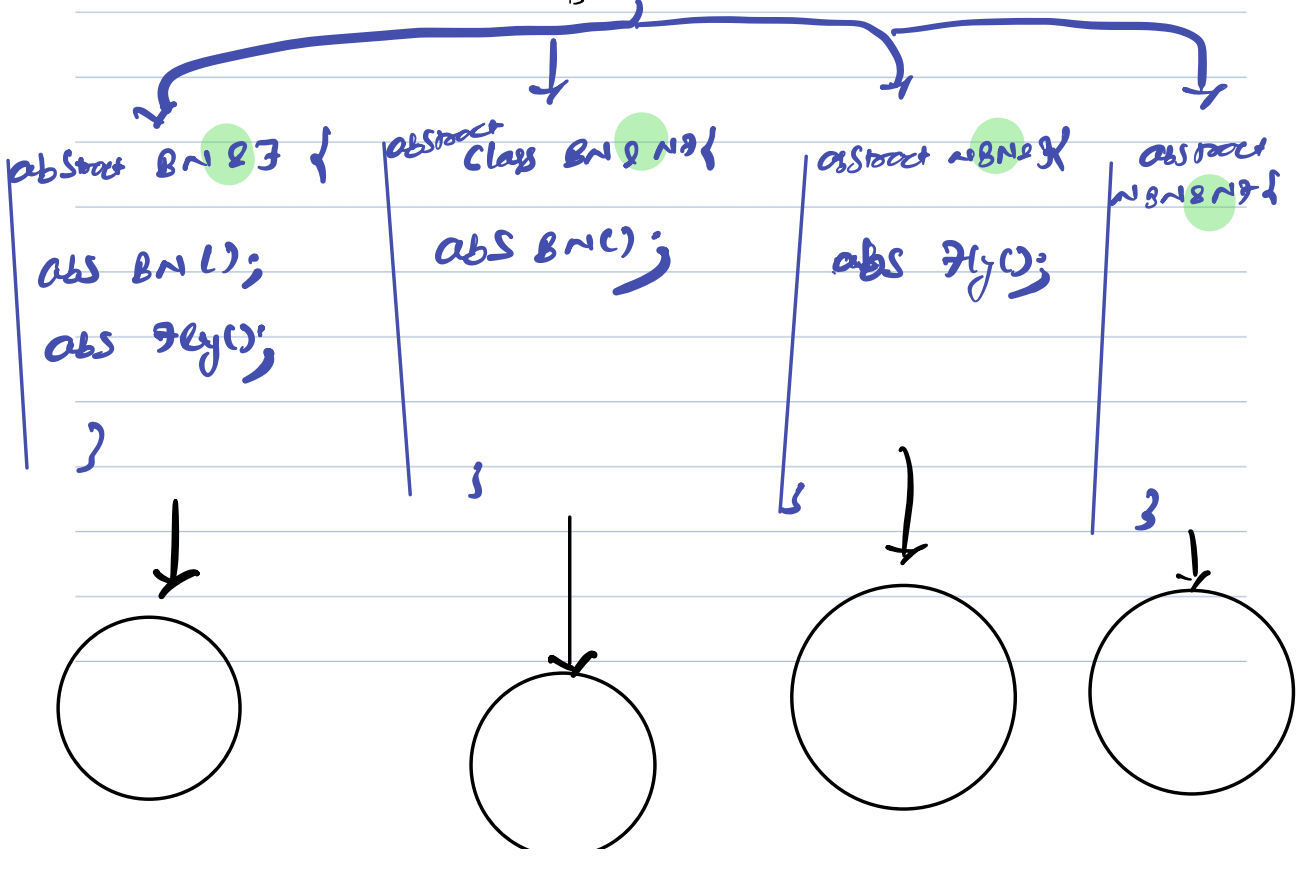


→ Some birds can build nest & some can't

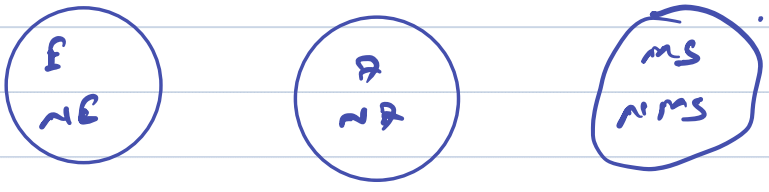


→ BN & F BN & NF NBN & F NBN & NF





→ Some birds can make sound, some can't?



2 * 2 * 2

↳ 8 combinations

2^N combinations

N = 10 → 2^{10} = 1024 possibilities

↳ find the need solution