Today's agenda

↳ SOLID Principle

AlgoPrep

* 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.

AlgoPrep

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

             Collect food();
             build nest();
             makeSound();
             fly();
}

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

Bird    b2 = new  Bird();
        b2. Color = white;
        b2. name = Pigeon";
        b2. weight = 300;
```

AlgoPrep

```
fly() {
   ≡
}
```

b1.fly()   b2.fly()

DRY

Parrot

~~[illegible]~~

```
void  fly() {

    if (name == crow) {
          ≡
    }
    else if (name == Pigeon) {
          ⋮
    }
    else if (        ) {

    }
    ⋮
    else if (crow) {
          ≡
    }
    else {
         ...
    }
}
```

→ How each type of bird should fly.

AlgoPrep

Problems:

↳ (i) Difficult to understand.

(ii) Extensibility difficult.

(iii) Difficult to test.

(iv) Code duplication

(v) merge conflict

$$\begin{tikzcd}
& \text{void fly ()} \arrow[dl] \arrow[dr] & \\
\text{dev1} & & \text{dev2} \\
\text{2 if else if} \arrow[dr] & & \text{2 if else if} \arrow[dl] \\
& \text{merge conflict} &
\end{tikzcd}$$

AlgoPrep

(vi) violates SRP {Single responsibility Principle}

(vii) violates OCP {open close Principle}

→ Single Responsibility Principle

↳ every code {method / class / Package}
must have exactly 1 defined responsibility.

↓

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

MVC

controllers / component
- ↳ user controllers
- ↳ group controllers

services
- ↳ user services

models
- ↳ Movie
- ↳ seat

IDentify violation of SRP

① Methods with multiple if-else.

exception

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

AlgoPrep

(11) Monster method → large

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

```
Savetodatabase ( . . . ){
    { calculate something . . . —
            — . —
              —

    { create db connection

      [ db.execute (query);

}
```

AlgoPrep

```
Savetodatabase ( -  . . - ){

        calculate ();
        createconnection();

    [ db. execute (data);
}
```

```
private int calculate ()(
    - -
    =
    - -
}
```

Integration solving

Reusable

```
private void createConnection()

}
```

③ Common / utils folder
↳ garbage place of your codebase.

→ Java.util
↳ fetch date
↳ create_map

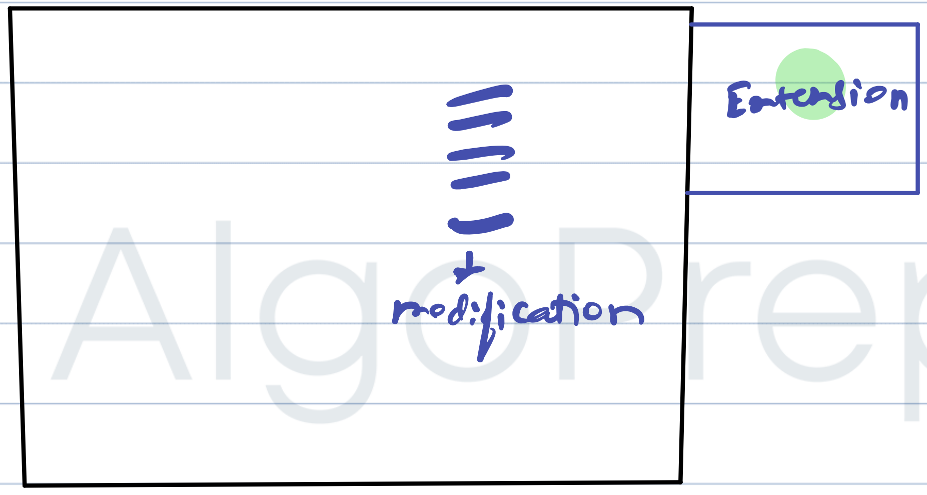
AlgoPrep

→ Open Close Principle

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

↓
easy to add new features

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



Break till 9:30 pm

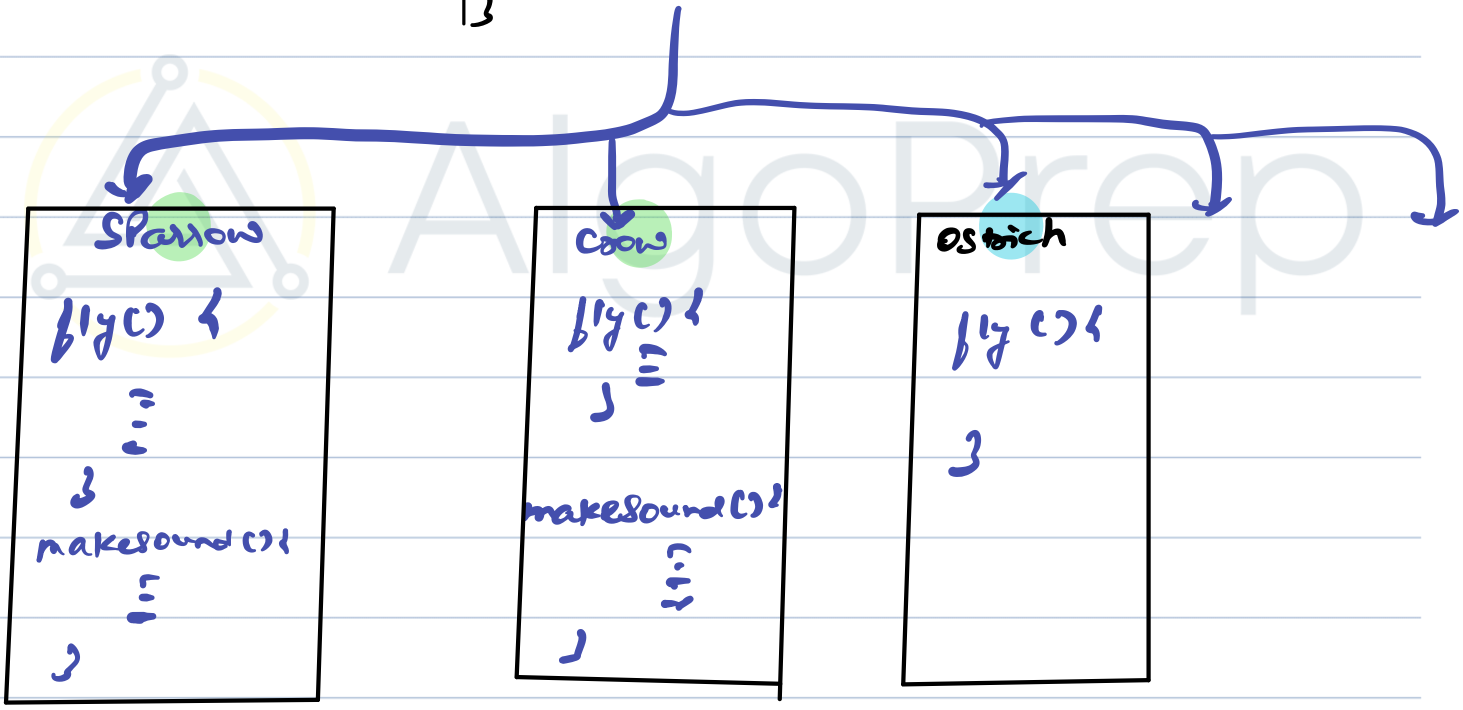
AlgoPrep

S1

```
abstract class Bird {
        Color;
        weight;
        name;
        age;
        breed;

    abstract collect food();
    abstract build nest();
    abstract makeSound();
        abstract fly();

}
```



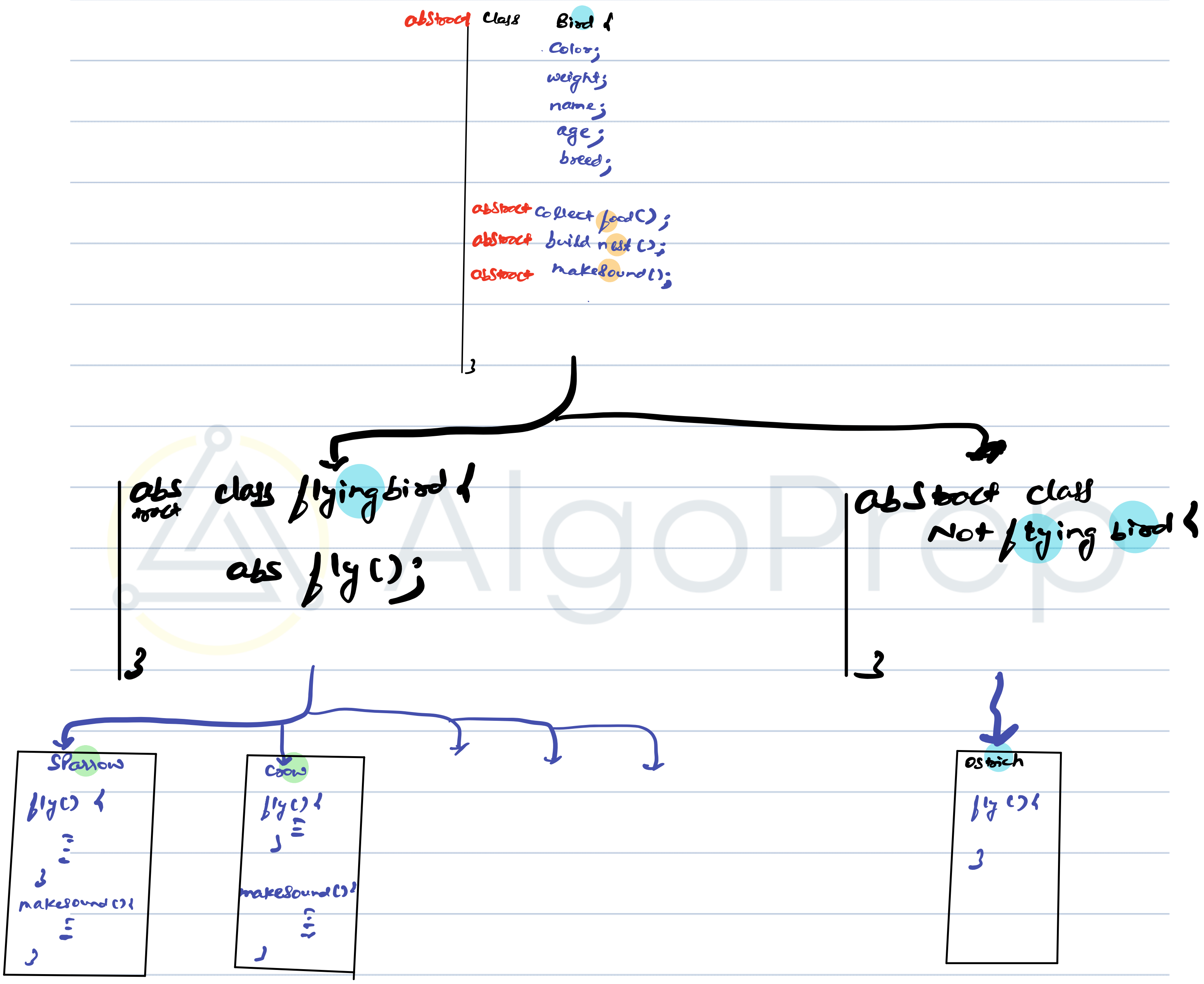
SRP ✓

OCP ✓

AlgoPrep

* Penguin, Ostrich

↳ birds which can't fly.



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

    abstract Collect food();
    abstract build nest ();
    abstract makeSound();
}
```

```
abstract class flyingbird {
    abs fly();
}
```

```
abstract class
    Not flying bird {
}
```

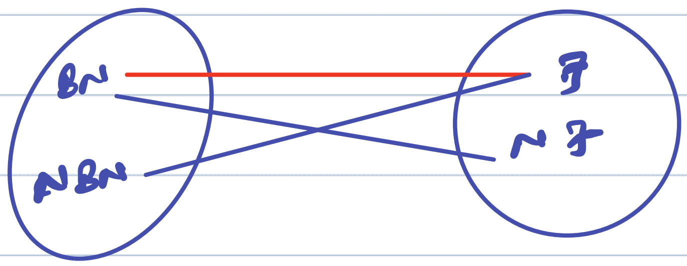
```
Sparrow
fly() {
    ⋮
}
makeSound() {
    ⋮
}
```

```
Crow
fly() {
    ⋮
}
makeSound() {
    ⋮
}
```

```
ostrich
fly () {
}
```

AlgoPrep

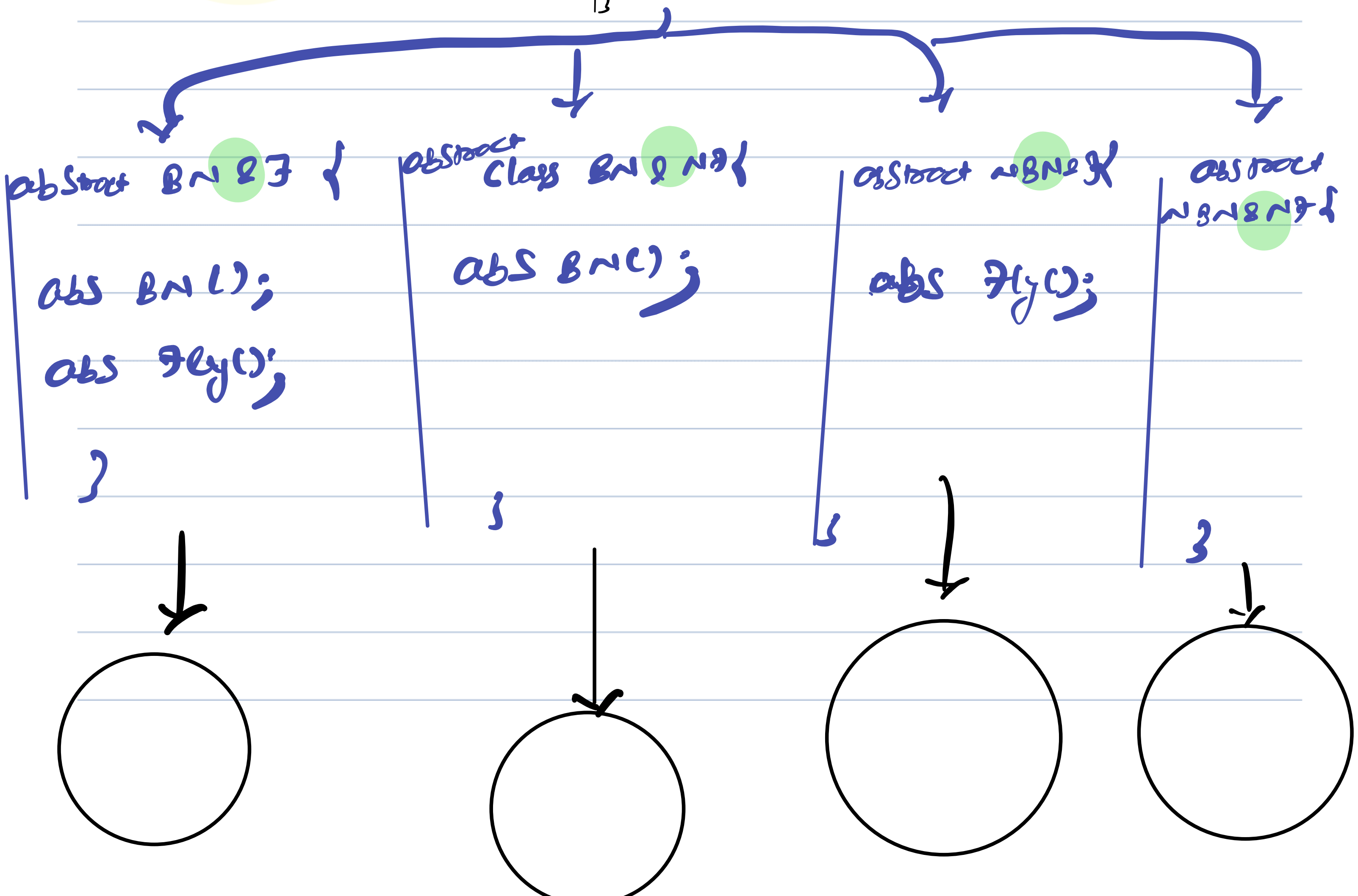
→ Some birds can buildnest & some can't



→ BN & F   BN & NF   NBN & F   NBN & NF

```
abstract class Bird {
    Color;
    weight;
    name;
    age;
    breed;

    abstract collectfood();
    ~~abstract buildnest();~~
    abstract makeSound();
}
```



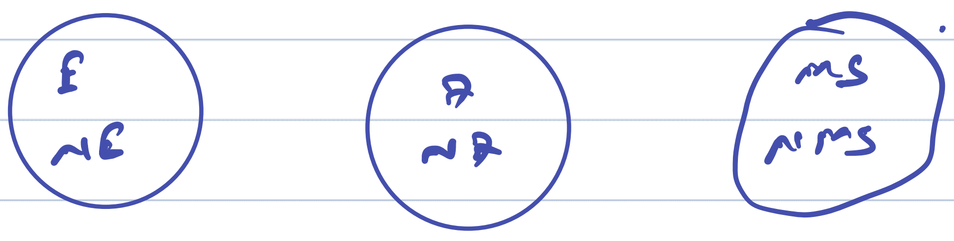
```
abstract BN & F {
    abs BN();
    abs fly();
}
```

```
abstract class BN & NF {
    abs BN();
}
```

```
abstract NBN & F {
    abs fly();
}
```

```
abstract NBN & NF {
}
```

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



$2 * 2 * 2$

↳ 8 combinations

2^N combinations

$N = 10 \rightarrow 2^{10} = 1024$ Possibilities

↳ find the next solution