Dr. APJ Kalam.

* Don't ask anyone till you yourself fail to find the ans.

"What if ---?"

→ Write down in notebook

→ Try it out (30 min) [write]

→ Do research
- google
- Blogs
- Stack overflow

Post in Slack

10-15%

85-90%

Maintenance
- Testing
- Found Bug
- Fix Bug

Understandability
- Understanding others code
- PR review.
- KT.

Extensibility
- Regression Bugs.
- Merge conflicts.

Easily
- Maintainable
- Understandable
- Extensible

```
int add(a, b) {
    ret a+b;
}

main() {
    print(add(2,3));
}
```

Functional

Procedural Programming

C

C++, Java

OOP

```
int length(Stirg s) {
    //
}
```

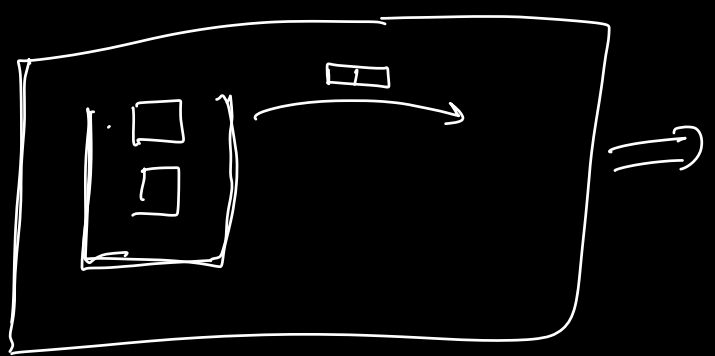
```
Class Stig {

    length() {
        //
    }

}

s.length();
```

Stat (Car c)



Amazon

- Orders
- Products
- Buyer
- Cart
- Seller
- Admin
- Add
- Im

* Design system representing a Bird.

→ Template / Blue Print.

Bird
- Hen
- Eagle

Bird
- → Properties
- → Actions / Behaviour

```
class Bird {
// Attributes
  double wt, ht;
  String color;
  type : → [hen, eagle]
// Methods
  void eat() {
    ⋮
  }
  void fly() {
    ⋮
  }
  ⋮
}
```

Data

Method

```
Bird hen = new Bird();
```
→ Constructor

```
hen.wt;
hen.color;
hen.fly();
```

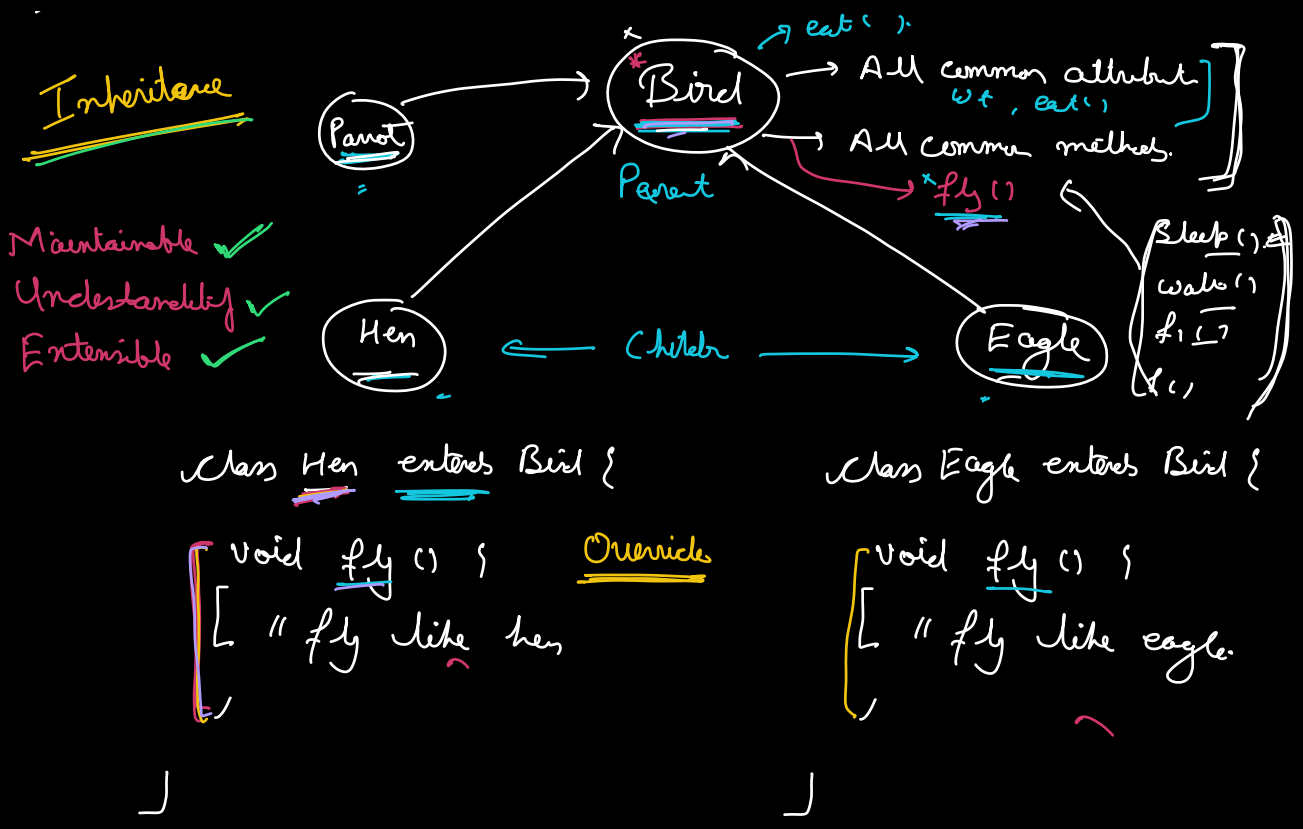
```
Bird eagle = new Bird();
eagle.fly();
```

```
void fly() {
  if(this.type == "hen") {
    // fly like hen
    [ ... ]
  }
  else if(this.type == "eagle") {
    // fly like an eagle.
  }
}
```

+SO if/else
→ +SO Behaviour

Maintainable ✗
Understandable ✗
Extensible ✗

+1 if/else



⇒ Make sure that no-one is able to create objects of the Bird class.

Bird
- All common attributes (wt, ht, cnt)
- All common methods (fly, eat)
- The implementation of all methods evolved are same for all birds (Sleep())

Bird → abstract Bird
- Which does not allow you to create objects.

⇒ fly method should be in the Bird class

But without implementation

```
fly() {
   =
}
```

→ `abstract fly();`

- No implementation is allowed in this class
- All the sub classes are forced to override it.

```
class Bird {
    abstract fly();
}
```

```
Bird b = new Bird();
b.fly();
```

No impl

Abstract method can only be in abstract class.

Break till 10:58 pm

```
main () {
    double a=2.5, b=1.5;
    Print (add (a, b));

    Str a="abc", b="xyz";
    Print (add (a, b));
```

add (_ _)
add (_ _)
add (_ _)

Method Overloading

```
int add (int a, int b) {
    ret a+b;
}
double add (double a, double b) {
    ret a+b;
}
string add (Sty a, Sty b) {
    ret a+b;
}
```

```
Class Angry Bird {
```

Abstraction

```
    Void render (Bird b) {
        b.fly();
    }
}
```

*fly()

⇒

Method Overloading

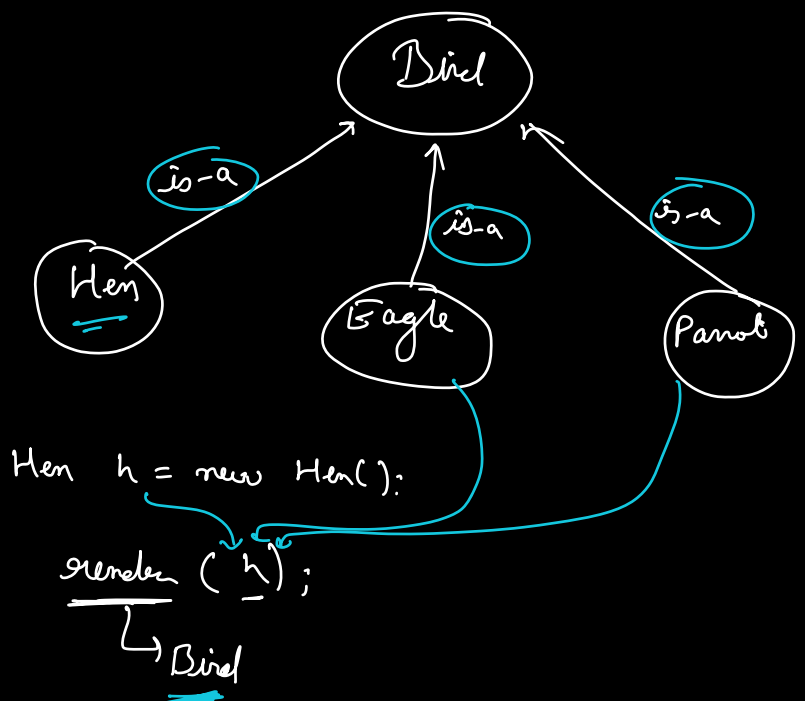
```
Void render (Hen h) {--}
Void render (Eagle e) {--}
Void render (Parrot p) {--}
```

Tight coupling

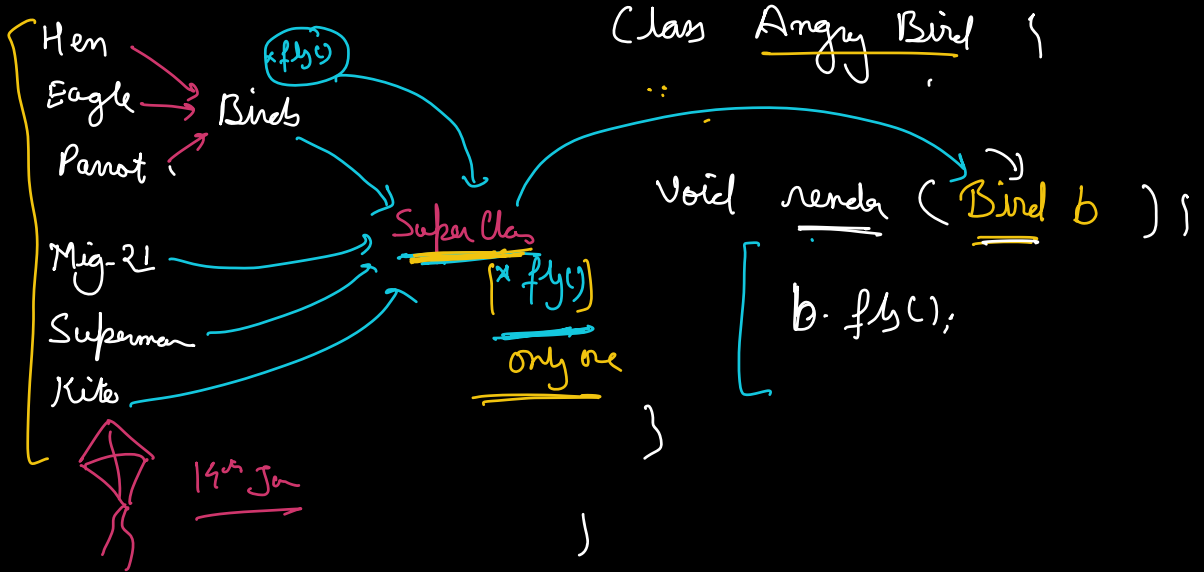
Client

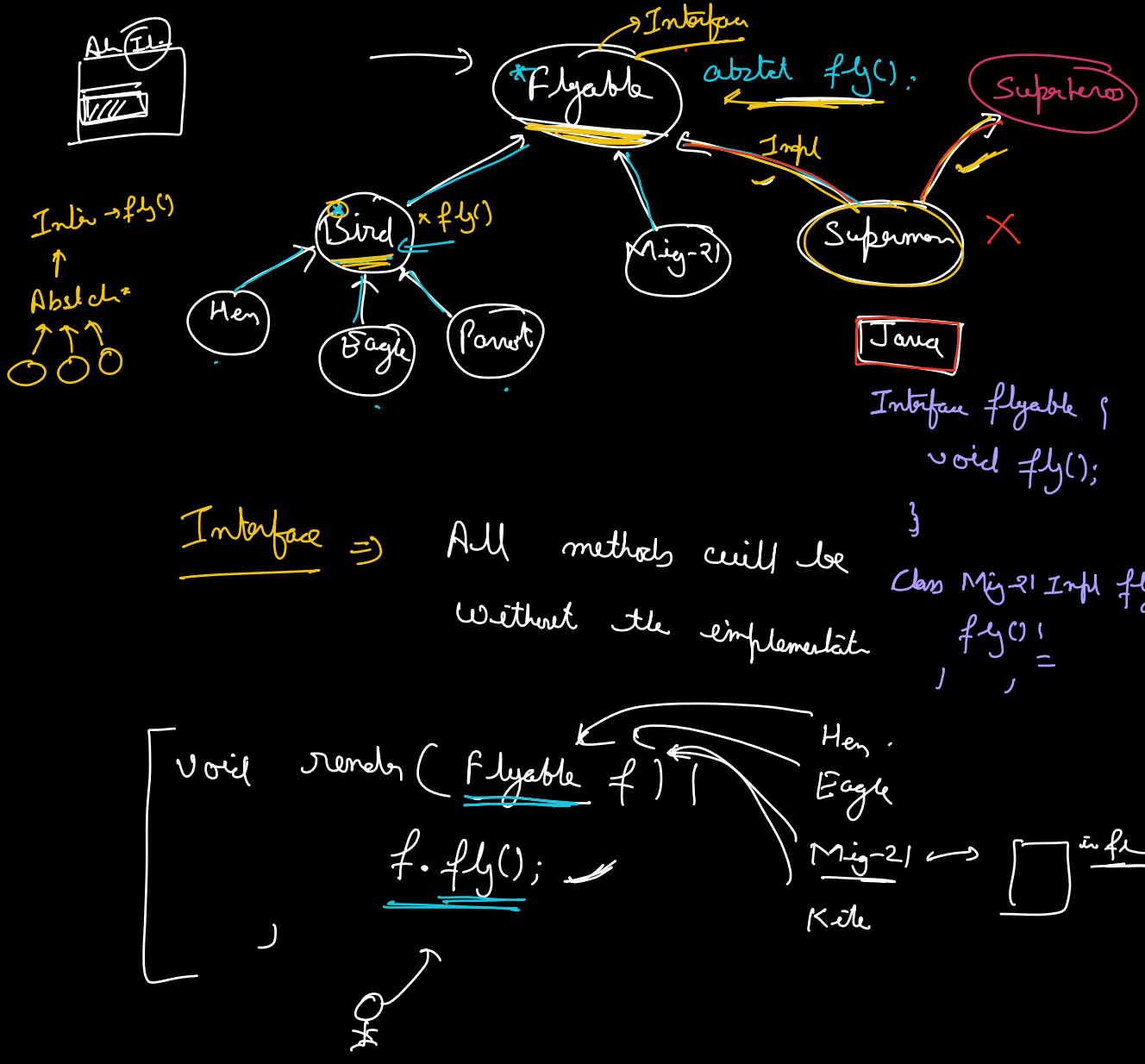
Bird
Hen
Eagle
Parrot

Virtual



Abstraction => Hiding the details.





Interface ⇒ All methods will be without the implementation

```
Interface Flyable {
    void fly();
}
Class Mig-21 Impl Flyable {
    fly() {
    }
}
```

```
void render(Flyable f) {
    f.fly();
}
```

Hen, Eagle, Mig-21, Kite → in fly

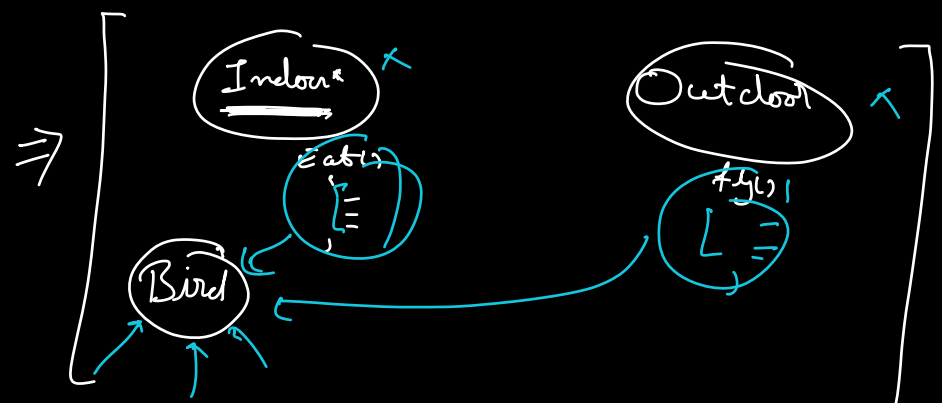
Many Form → Poly morphism

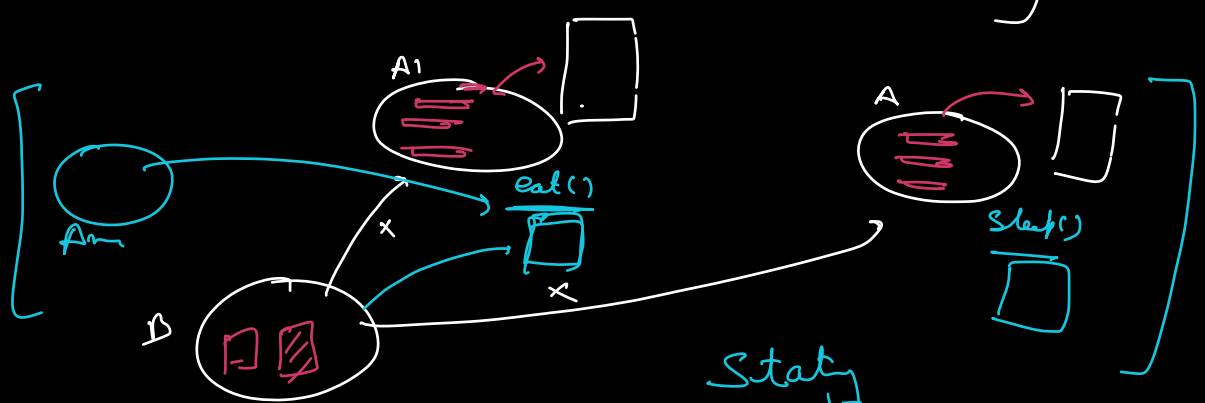
- Method overloading
- Method overriding

- Inheritance ✓
- Abstraction ✓
- Polymorphism ✓

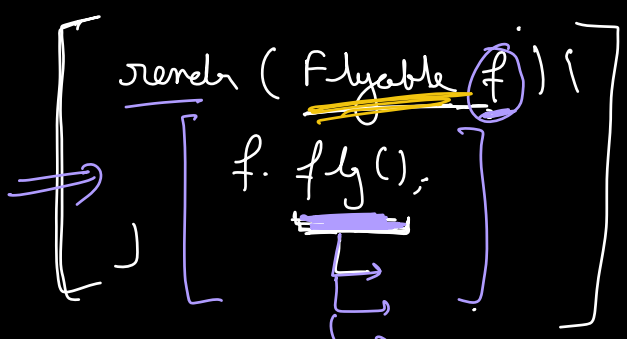
- → OOPs
- → SOLID
- → Design Patterns
- → LLD of 5-8 systems.
- → Machine coding
- → Database schema design

15 classes



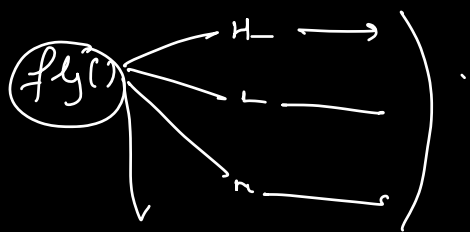


Static



Bird → Flyable

May-21 →


fy()
H
L
n