"Don't ask anyone till you yourself fail to find the answer"
– Dr. Kalam

"What if... ?"

→ Write down the question
→ Write code & observe behavior
→ Find the ans of why ?
- Google
  - Stack Overflow
  - Github
  - Blogs
- Peers
- Instructor / TA / Mentor

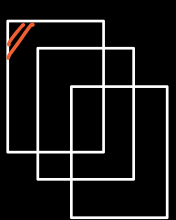
10-15%

85-90%

Maintainable
- Testing
- Find Bugs
- Fix Bugs
- Refactor

Understandable
- Understanding other code
- Code Review
- KT (Knowledge transfer session)

Extensible
- Regression bugs
- Merge Conflicts



1.5 months

- OOP (Inheritance, Abstraction, Polymorphism, Encapsulation)
- SOLID
- Design Patterns
- UML

- Case Studies
  - LLD
  - Code
  - Schema design

SDE-2 Amazon

Q Give a object oriented design of a Bird.

Must have

→ eat()
→ fly() "All birds must fly"
→ walk()
→ chirp()

Bird
→ Properties
→ Action

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

```
hen.fly();
```

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

All birds must have a diff flying behaviour.

```
class Bird {                    → Template / Blue print
  // Attribute
  String color, type;
  double wt, ht;

  // Method
  void eat() {
    // - - - -
  }
  void chirp() {
    // - - -
  }
  void fly() {
    // - - -
  }
}
```

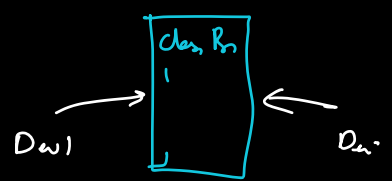
```
fly() {
  ---
  if (this.type == "hen") {
        // fly like a hen } -> algo1
  }
  else if (this.type == "eagle") {
        // fly like an eagle } -> algo2
  }
```

fn() {

LGTM

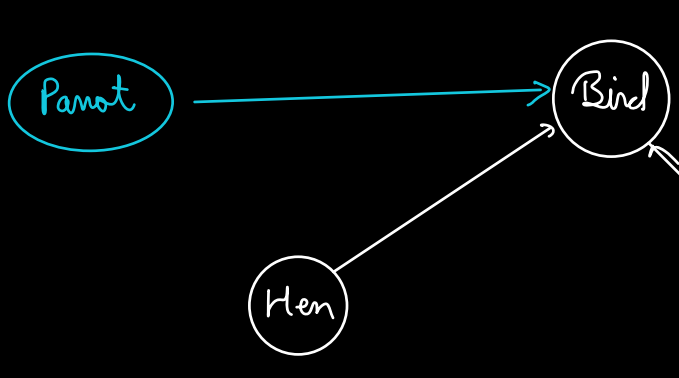
So if/else

So algos



Single Responsibility Principle

Every class / method must have only one responsibility.



String color, type;
double wt, ht;
fly() {
eat()
;

Class Hen extend Bird {
// Override
Void fly() {
// fly like a hen
}
}

Class Eagle extend Bird {
// Override
Void fly() {
// fly like an eagle
}
}

Class Bird {
fly() {
}
}

Bird b = new Bird();

b.fly();

X

Angry Bird

Class Bird {
fly() {
// common behavior
- fly like a bird
}
}

Class Flamingo extends Bird {

}

Flamingo f = new Flamingo();
f.fly();

abstract method

abstract class ⟶ A class whose objects can not be created.

abstract method ⟶
- No implementation in parent class
- Can only be present in an abstract class
- All children classes must implement the method.

```
abstract class Bird {

    String color, type;
    double wt, ht;

    void eat() {
        // implementation
    }

    abstract void fly();

}
```

Class Angry Bird Game {

void render ( ) { → Render the bird
→ make it fly

Method Overloading

void render (Hen h) {
h.fly();
}

void render (Eagle e) {
e.fly();
}

⋮

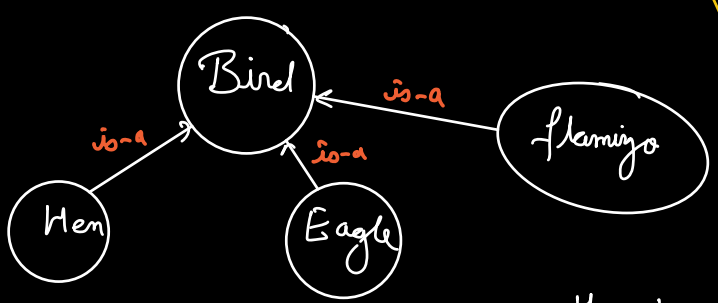
How many birds ?
New bird ?
Removed from birds ?

int add(int a, int b)
int add(double a, double b)
Polymorphism
(Compile time)

Abstraction

}

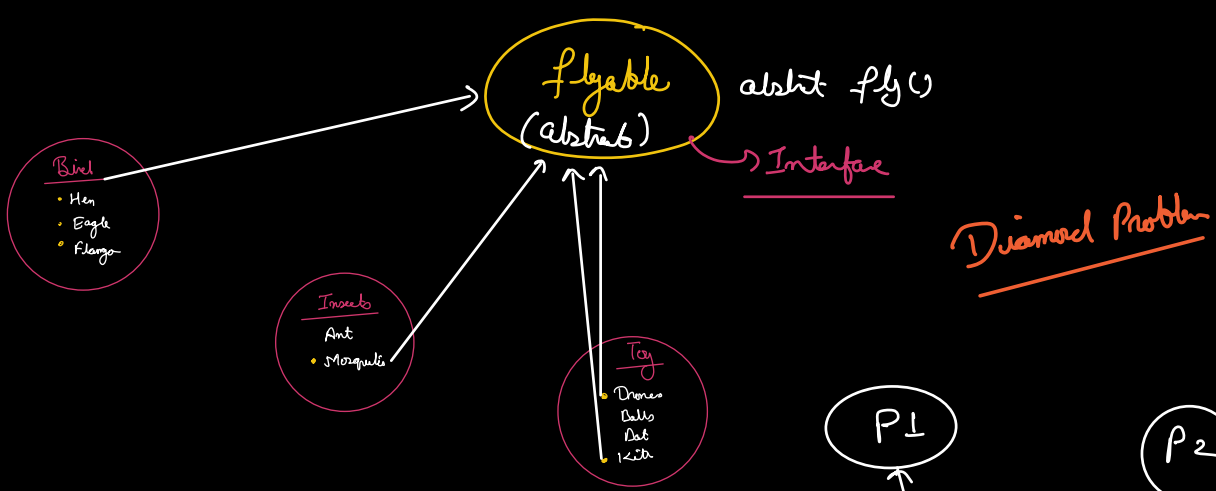
void render (Bird b) {
b.fly();
}



Polymorphism
many form

Hen h = new Hen();
render(h);

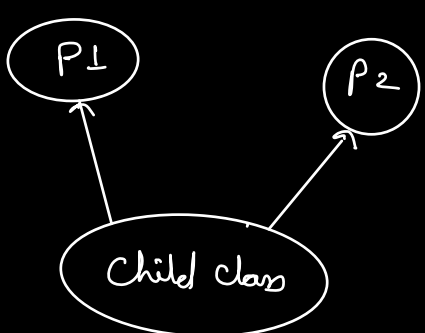
```
Class FlyingObjectGame {

    void render (Flyable f)
        f.fly();  ——→ call fly method
    }
}
```



Diamond Problem

```
Class P1 {
    void f1() {
    }
}
```

```
Class P2 {
    void f1() {
    }
}
```




```
CC c = new CC();
c.f1();
```

Interface ⟶ Pure abstract class

```
interface Flyable {
  void fly();
}
```