Agenda

S O L I D

Liskov's substitution Principle

Object of any child should be as-is substitutable in a variable of parent type, without requiring any change.

```
Bird b = new Penguin();
         new Pigeon();
         new Crow();

func(Bird b) {

    b.fly()

}
```

new Pigeon()
new Sparrow()

X new Penguin()

b.fly() → throws exception

Interface Segregation:

1. Interface should be as light as possible
2. As less method as possible
3. Ideally it should have only one method.

- Some birds can fly
- Some bird can dance
- All who can fly can dance

① <<Fly>> <<Dance>>

② <<FlyDance>> ✓ ✓
<<Dance>>

1st approach is better.

clear code

<<Stack>>
pop
Peek
push

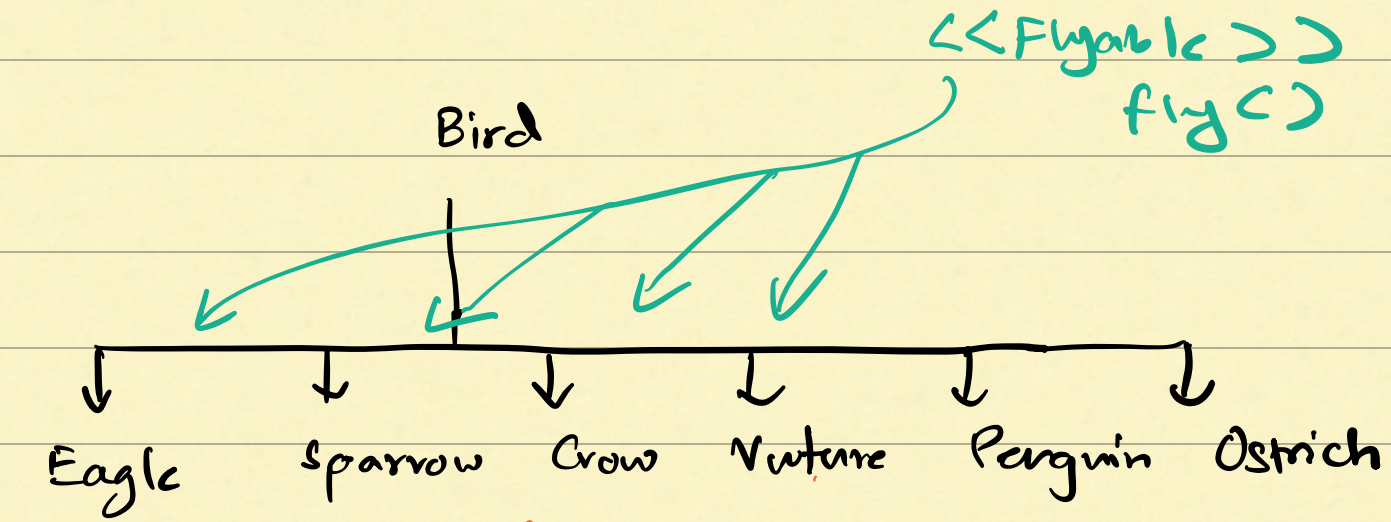
<<Popable>>
pop

<<Peekable>>
peek

<<Pushable>>
push

→ Abstract / class : Entity
Interface : Behaviour

Dependency Invasion



fly(){ } fly(){ } fly(){ } fly(){ }

Code Duplication

flyLow(){ }

flyHigh(){ }

class FlyLow {
int speed;
doFly(){ }
}

class FlyHigh {
int speed
makeFly(){ }
}

class Eagle {
FlyHigh f = new FlyHigh( );
fly(){ }
}

FlyLow

FlyLow()

```
        f.makeFly();
              ↳ doFly()
    }
}

Interface FlyingMethod {
    void makeFly();
}

class FlyLow implements FlyingMethod {
    int speed;
    makeFly() {
    }
}

class Eagle {
    FlyingMethod f = new FlyLow();
    fly() {
        f.makeFly();
    }
}
```

FlyHigh();

Dependency Inversion

↳

Two concrete class should not

directly depend on each other.

Eagle → FlyHigh()
FlyLow()

Eagle → (I) FlyingMethod

FlyHigh ↑ FlyLow ↑

Uncle Bob → FRAGILE

SOLID → Michael Feathers.

Positive.