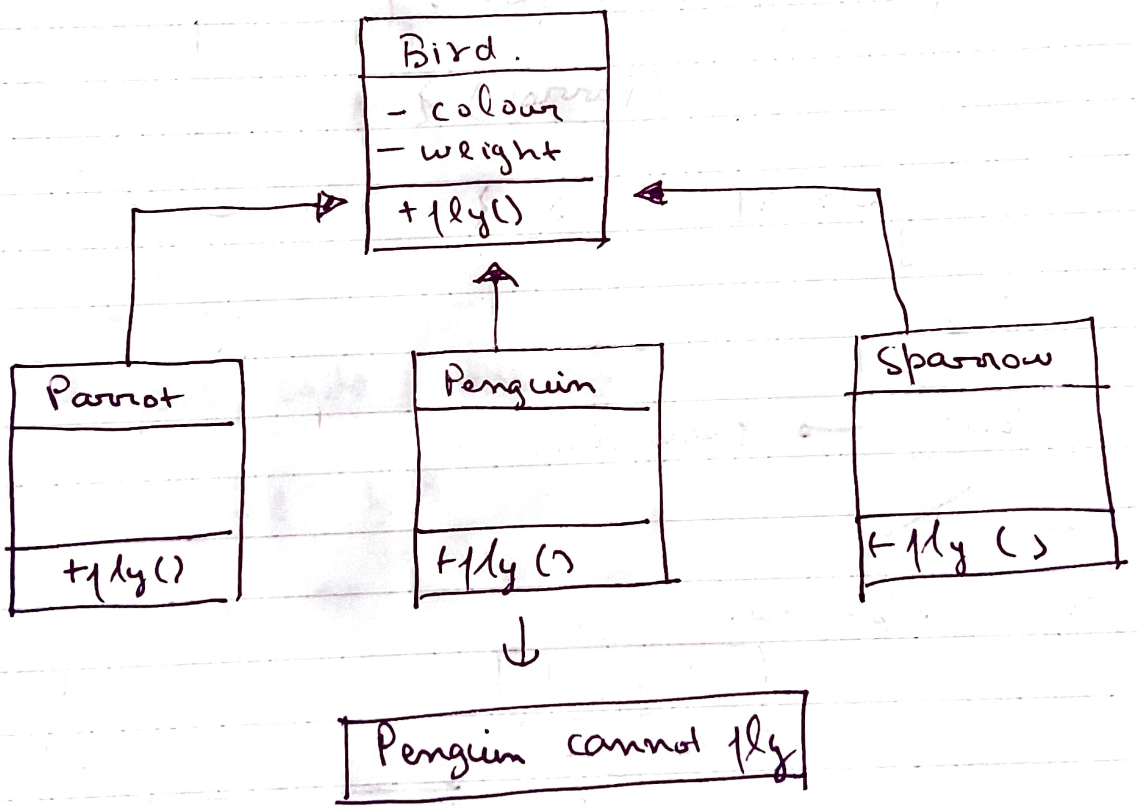
Day 06 Solid Principles

Liskov's, Interface Segreation, Dependency Inversion

1. Liskov subsitution principle.
2. Interface segreation.
3. Depandency Inversion.



```
fly() {


}.
```

① return null }
② throw an exception.
③ Dummy method

Release all birds

```
release (List of Birds) {
        List< Bird
    for (Bird b in Birds) {
        if (!b instance of Pegoin){.
            State = b.fly()
        if ( state !null && State == FLYING){
                b. make sound ()
        }
}
```

→ special handling for a child class.

→ substitute a child as a parent
↓
subtype.

→ If you have to handle a special class
↓
LSP ×

LSP ☆

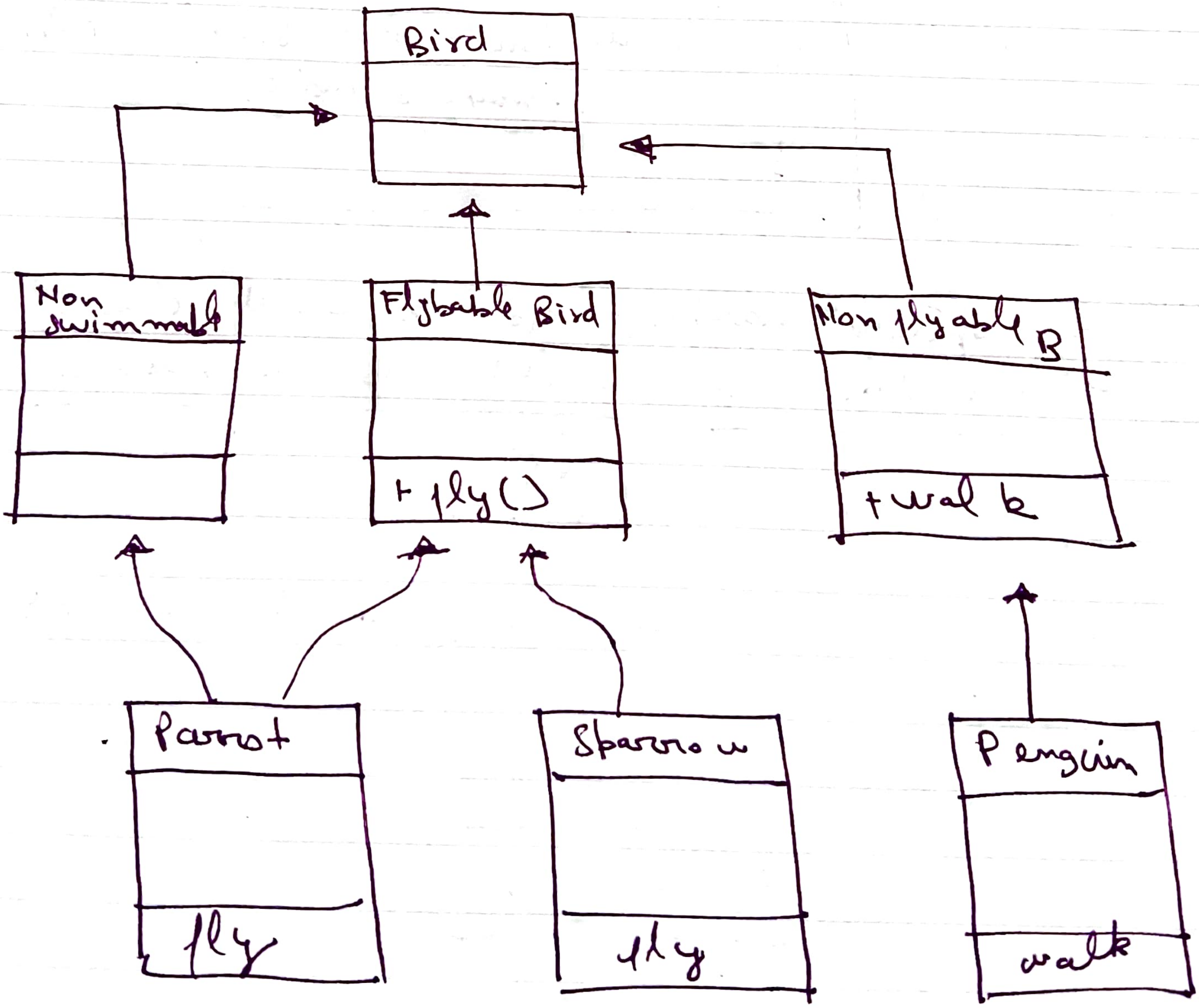
Object should be replacable by their sub types

without altering the correctness of the program

→ How can we fix LSP?

① Default impl
↳ still implement fly.

② Multilevel inheritance



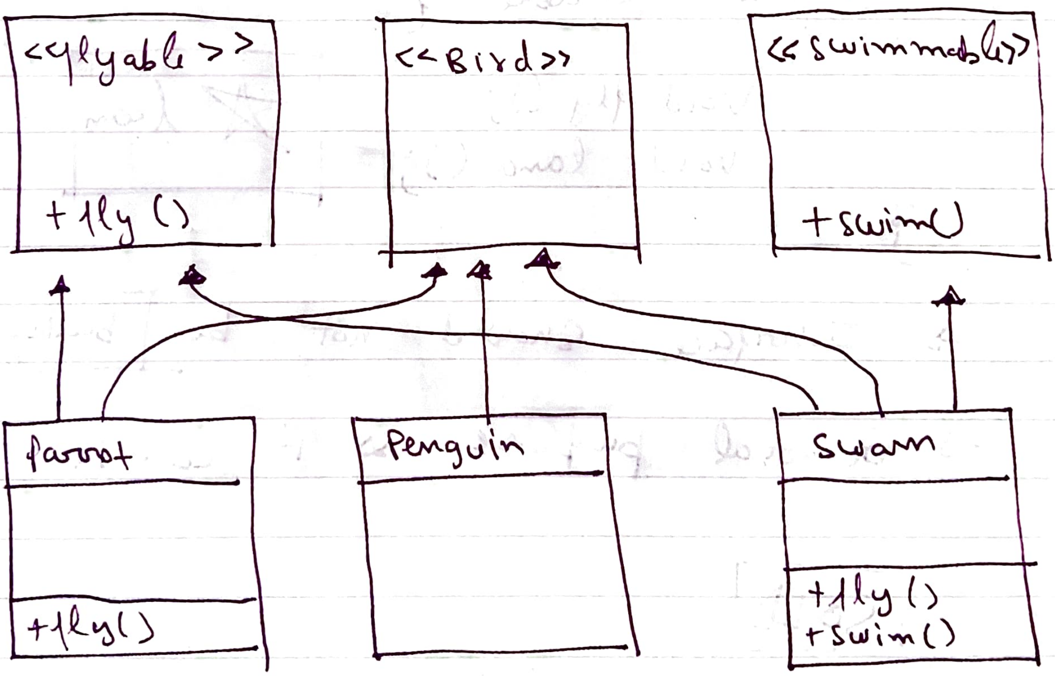
release (List < > birds)

for all
b.fly() Flyable Bird

→ Swim
- → Parrot cannot swim.
- → Penguin can swim.

Tying behaviour with inheritance hirarchy | X

Interfaces



① Replacable with subtypes.

② To check → Special cases, except some behaviour.

③ Perfer using interfaces to abstract.

Do not tie behaviour with hierarchy

Interface Segregation.

↓

Separate.

→ creating lean interfaces

```
interface flyable {

        void fly ();
        void land ();
}
```

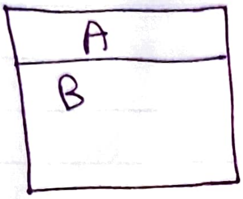
★ lean

→ Interface should not be **bulky**

→ General purpose so it can be reused

SOLD

Dependency Inversion

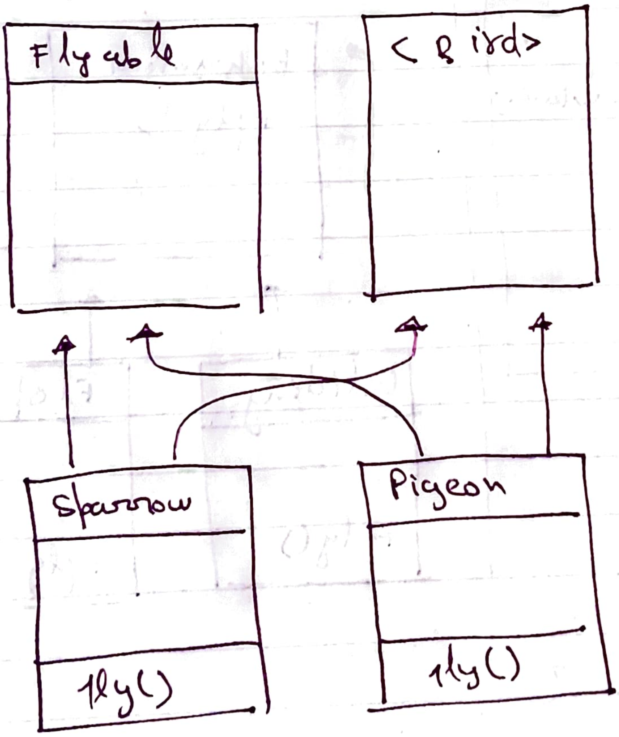


A → B.

new A(new B())

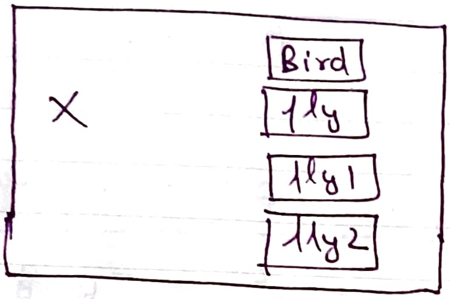
```
class Pigeon extends Bird implemants Flyable {

}.
```

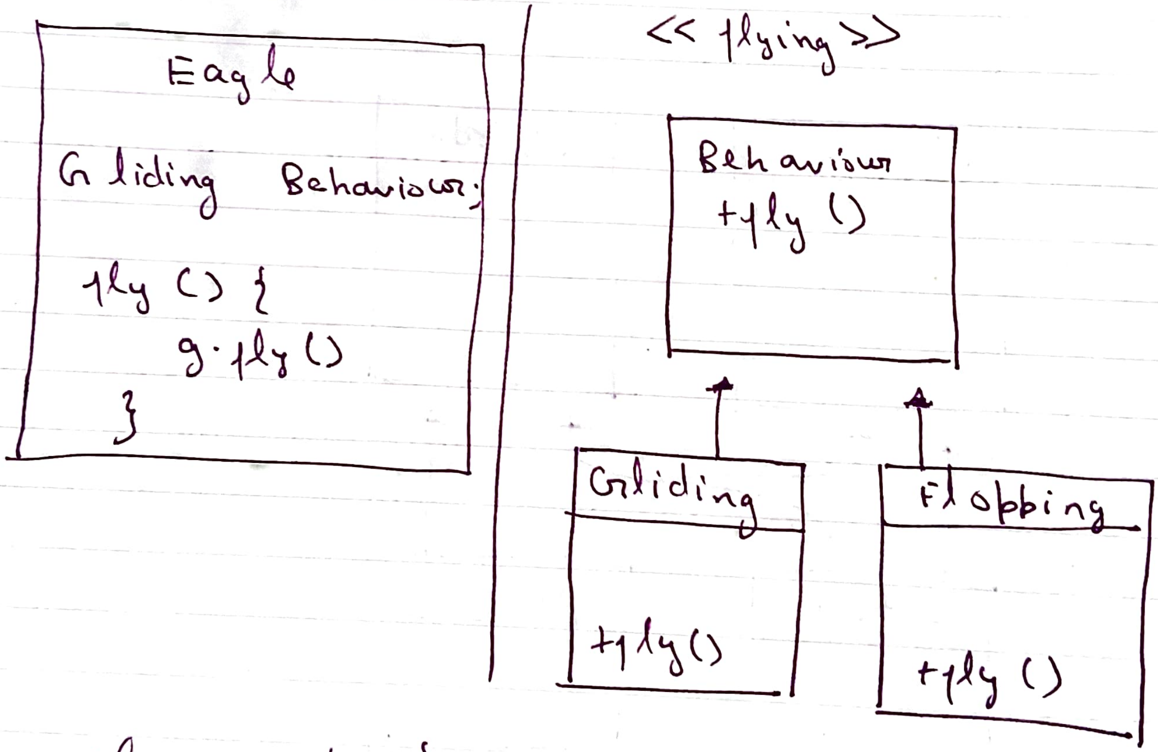


Solutions

1. Default imp — Hand to decide default.



2. Utility method



```
class Edge {


}.
```

Dependency inversion.

↳ Concrete classes should not depend on other concrete classes

↓ Sparrow

↓ Gliding Behaviour

↳ Depend on abstractions

↓ Flying Behaviour

loose coupling

Code Unit

SRP → One class one responsibility.

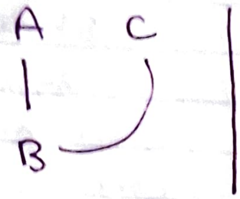
OCP → closed for modifications
open for extensions.

LSP → Do not enforce behaviour on child classes
→ If subtyping, all behaviour should work as expected.

ISP → lean interfaces.

DIP → concrete classes should spend on interfaces rather than each other.

→ Inheritance is not dynamic.



A

B.

$\boxed{\text{Composite is dynamic}}$

A · method ()

```
class A {
    B b
    C c
  Static method () {
  }
}
```

```
class X {

   method ()
}.
```

Ⓐ · method ()

$\boxed{\text{new A() · method ()}}$

```
class Manager {
  Db db;
   execute () {

      db . execute(Qu.cq())
   }
}
```

```
class query {
      Db db
   Static createq() {

      }
}
```