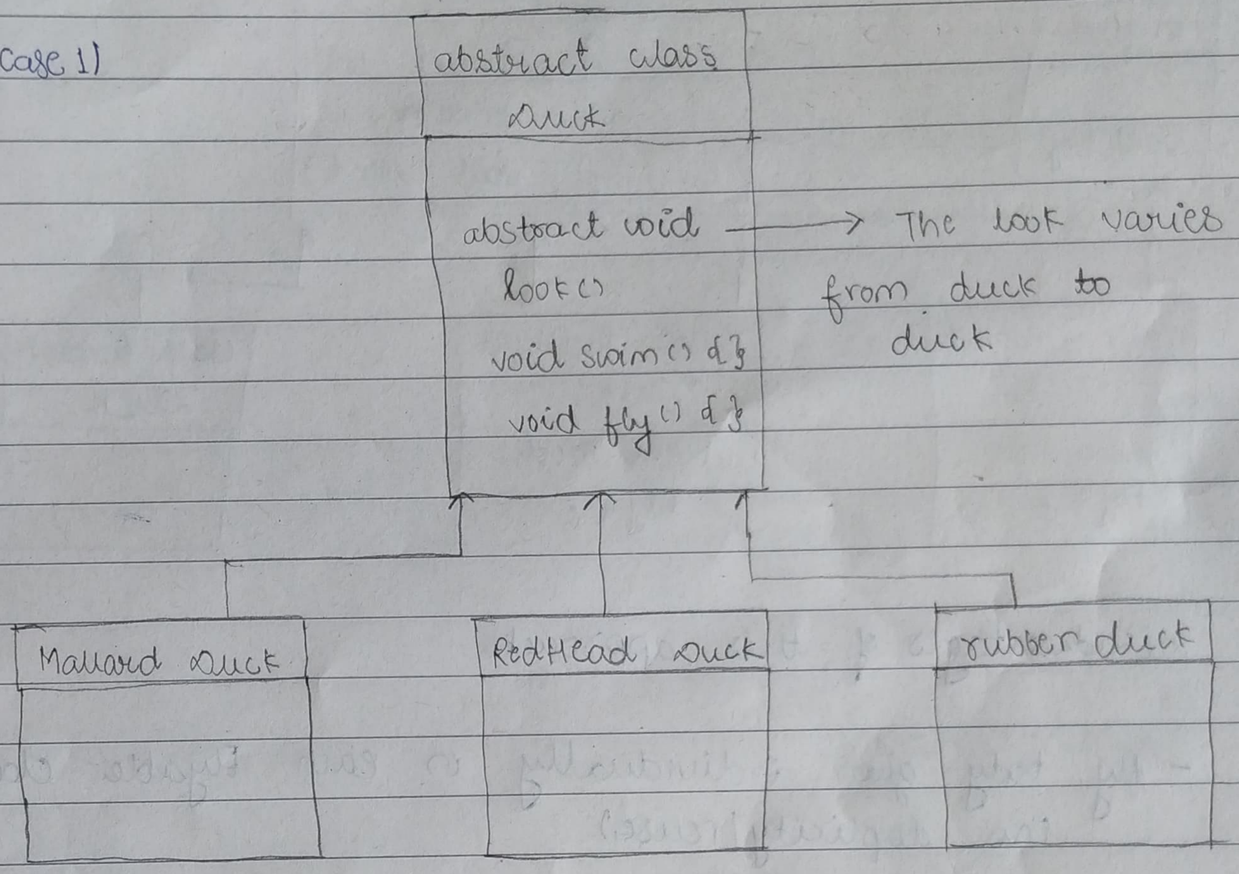
The Strategy Pattern

- suppose, I'm simulating a 'duck' video game.

case 1)



wait a minute but rubber ducks are not supposed to inherit fly method.

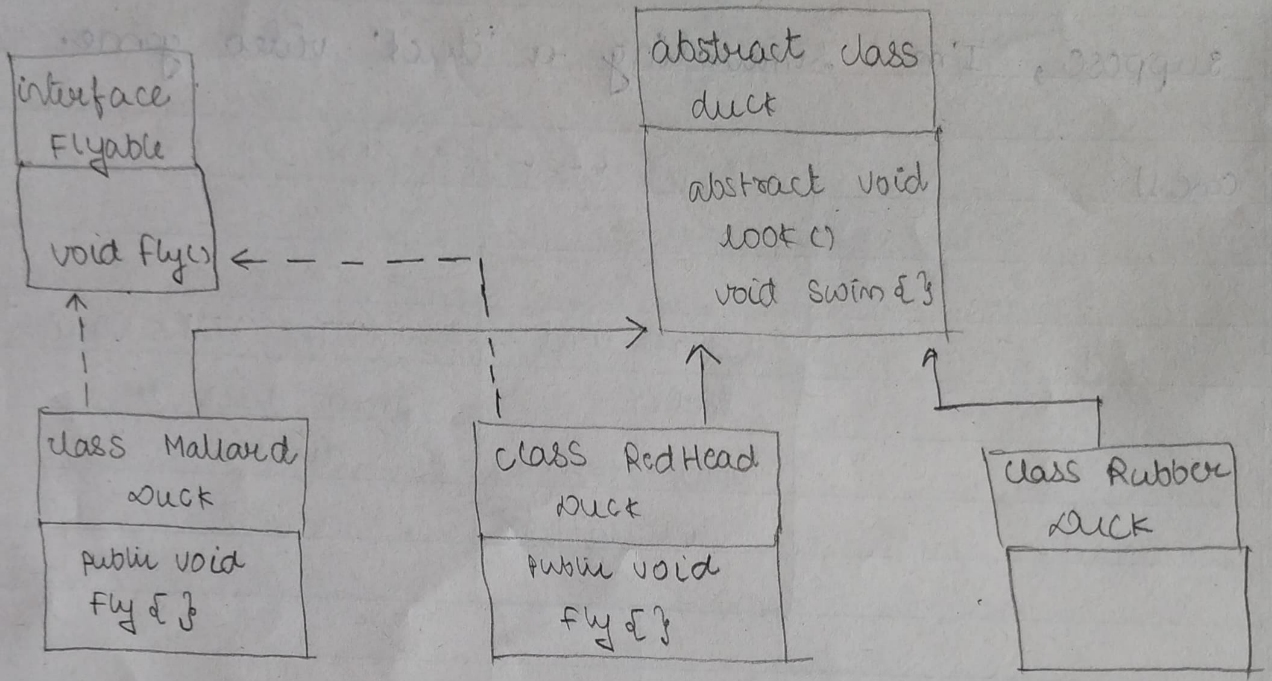
Design principle →

Identify ascpets of your applications that vary (fly method()) and seperate them from what stays same.

Encapsulate what changes so it doesn't affect rest of code.

case 2)

so, you tried an interface & implementing interface Flyable only those classes, which can fly.



disadvantages of this approach

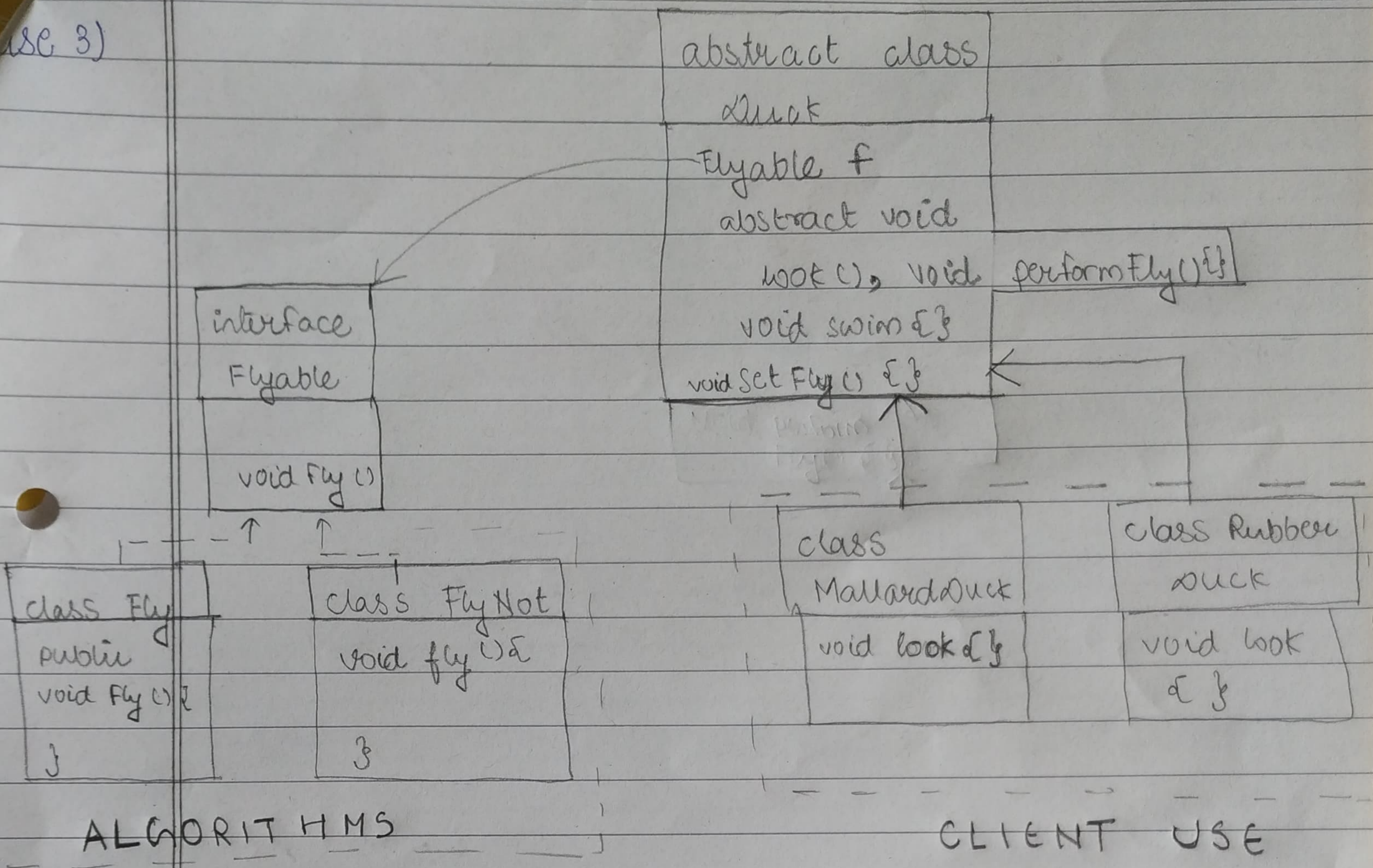
- fly body given individually in each Flyable class (no duplicity/reuse)

- To change behavior of fly method touch all Flyable subclasses.

- Touching subclasses might bring lot of bugs.

This approach is
- inflexible
- makes maintainence difficult.

ase 3)



design principles

- Program to an interface, not inplmentation.
- Prefer 'HAS-A' over 'IS-A'.

* 'DEFINITION'

strategy pattern defines a family of algorithms, encapsulates each one & makes them interchangable.

It lets algorithm vary independently from clients that use it.