BIG O

South Africa carrier pigeon VS Internet

- Well the carrier pigeon has a relatively constant transport speed, regardless of the amount of data it has to transfer.
- The internet, due to it's nature, takes longer to transfer data, ~~the~~ depending on the amount of data there is to transfer.

Pigeon = 30 m/h regardless of data

WEB = 100 mb/s

So, if there's a need to transfer data over the course of 30 miles. Let's say it's 100 GB of data, of course the pigeon will be faster

ν = constant time. $O(1)$

web  = linear time, with respect to the amount of input. $O(N)$

Big O -> how time scales with respect to some input variables

Pseudo code

```
boolean contains(array, x){
  for each element in array {
    if element == x {
      return true
    }
  }
}
```

Linear

$O(N)$

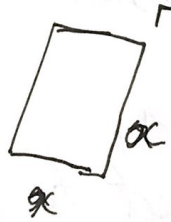
↳ where N is the size of the array

```
void printPairs(array){
  for each x in array {
    for each y in array {
      print x, y
    }
  }
}
```

$O(N^2)$

↳ squared, because it's the same input twice

$\sum_{n \geq 1} N \times N$



→ how much time would it take to ~~fill~~ color this square with

$O(a)$? => a -> area $= x^2$

therefore it's quadrite

· The variables and their meanings are important

4 Rules for big O

1. Different steps get added
2. Drop constants
3. Different inputs => different variables
4. Drop non-dominate terms

1.

```
function doSomething() {
        step 1();  // O(a)
        step 2();  // O(b)
}
```

$O(a+b)$

2.

```
function minMax(array){
    min, max = NULL
    for each e in array{
        min = Min(e, min)
    } for each e in array{
        max = Max(e, max)
    }
```

$O(N)$ $O(N)$ $O(N)$ not $O(2N)$

we do this, because we're not trying to check exactly how long, but how it computes (linear)

```
function minMax(array){
    min, max = NULL
    for each e in array{
        min = Min(e, min)
        max = Max(e, max)
}
```

$O(N)$

③

```
int intersectionSize(array A, array B) {
    int count = 0;
    for a in array A {
        for b in array B {
            if a == b {
                count += 1
            }
        }
    }
    return count
}
```

different sized arrays

$O(a \times b)$

④ if in a function there's a loop that's linear and another that's squared, we don't do this:

$O(N^2) \leq O(N + N^2) \leq O(N^2 + N^2)$

N^2 is the dominant term.

For the same reasons as in rule two, we only care about $O(N^2)$

ABSTRACT CLASSES

A class that represents a generalization and provides functionality, but it's only intended to be extended and not instantiated. "Super class"

Should not be instanciated

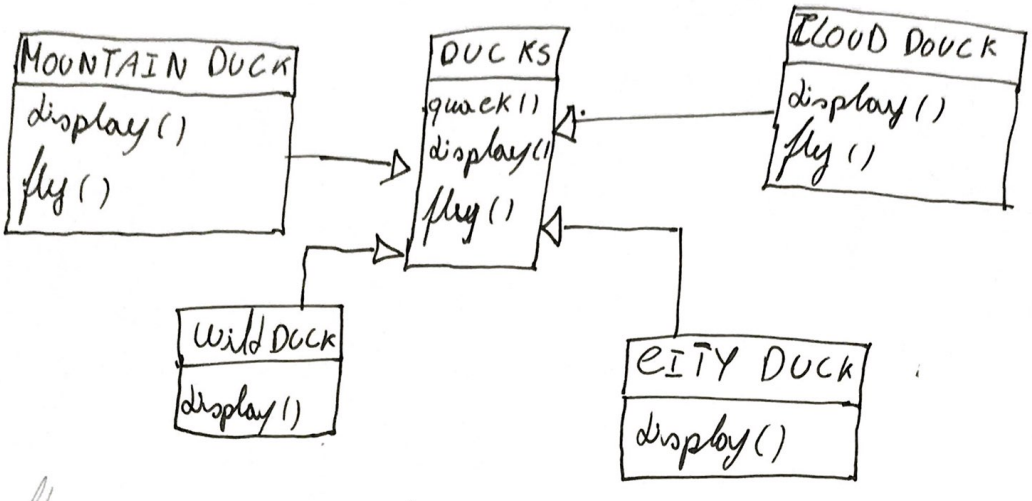
- May contain a constructor, but it can never be called since the class will never be instatiled. No "new"
- May contain abstract methods
- Any class that contains an abstract method is itself an abstract class regardless of how it is defined.

INTERFACE

A completely abstract class that defines a protocol for object interactions

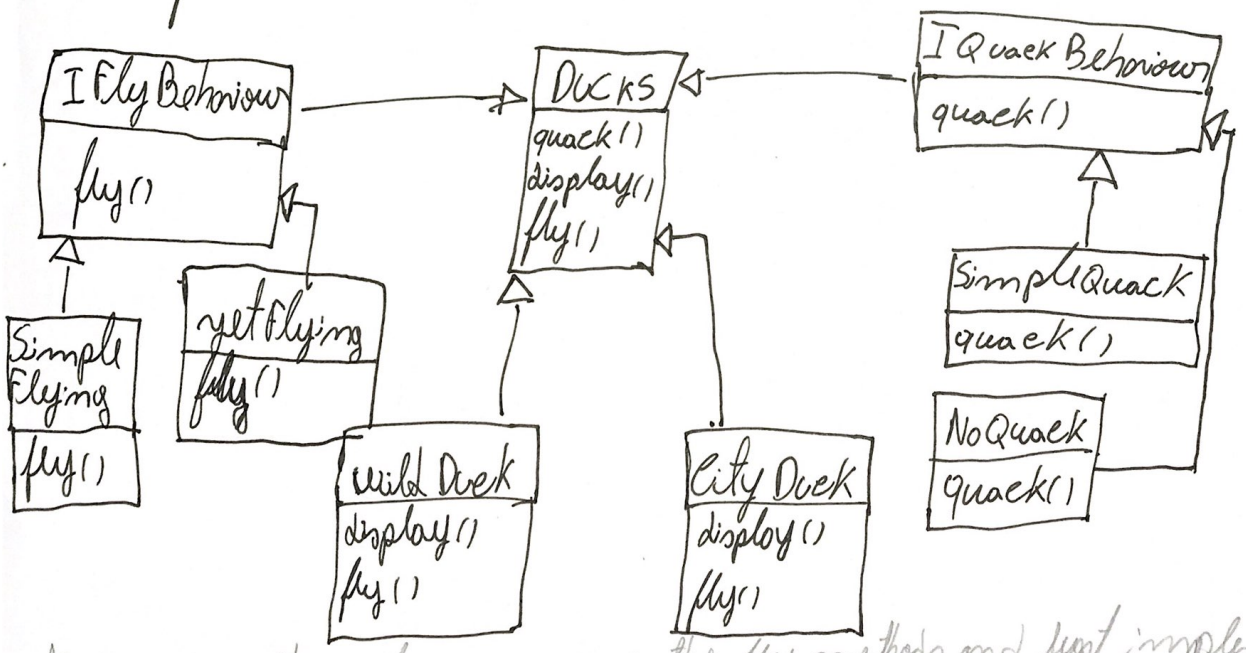
- May contain only static final variables
- May contain only abstract methods
- Can not contain a constructor as interfaces can not be instantiated
- Interfaces can extend other interfaces
- A class can implement any number of interfaces
- A class that implements an interface has <u>is-a</u> relationship with that data type

STRATEGY PATTERN



- For this example let's say that, the display method has a different implementation for each class, and that the fly method is equal for each class it's on. This is ineffective, since the DUCKS class is an abstract class, we will basically have to copy + Past the fly method on to other classes.

what if:



- Now we can choose from among the fly methods and just implement them. This allows for more code reusability.