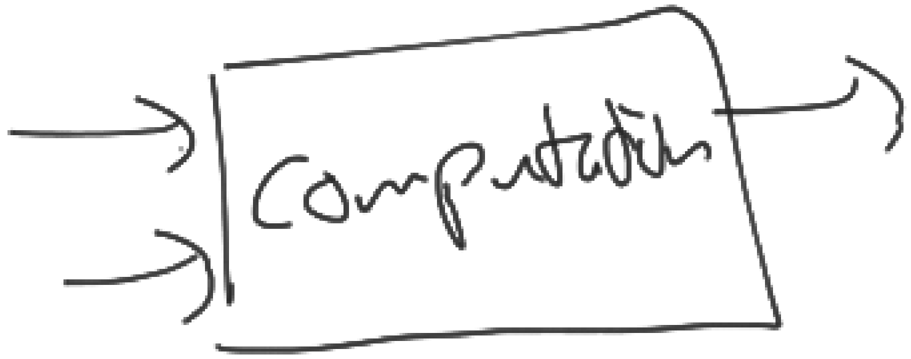
Abstraction



function

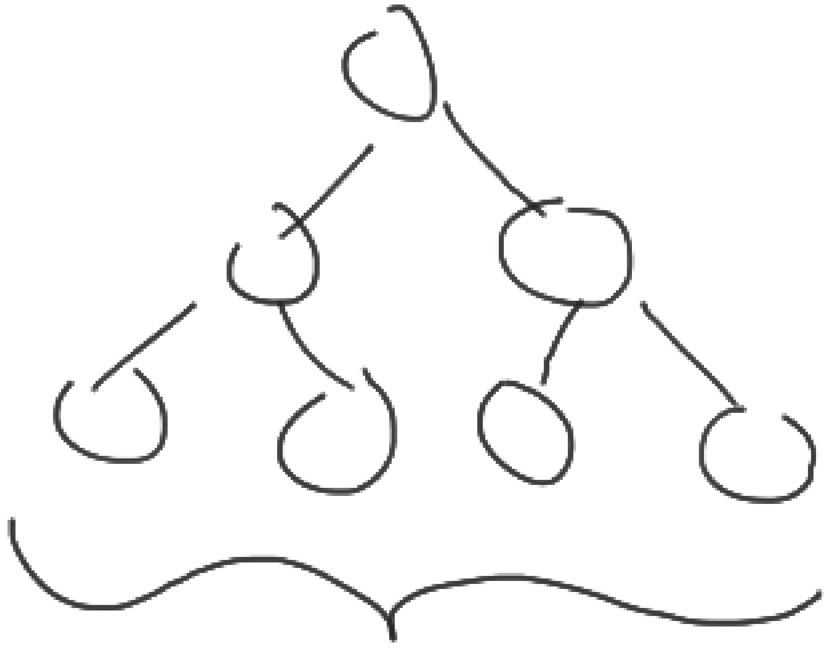
Data + Compution

int, flat, str
list, dict, tuple, set

Recursion

→ Divide & Conquer

heap



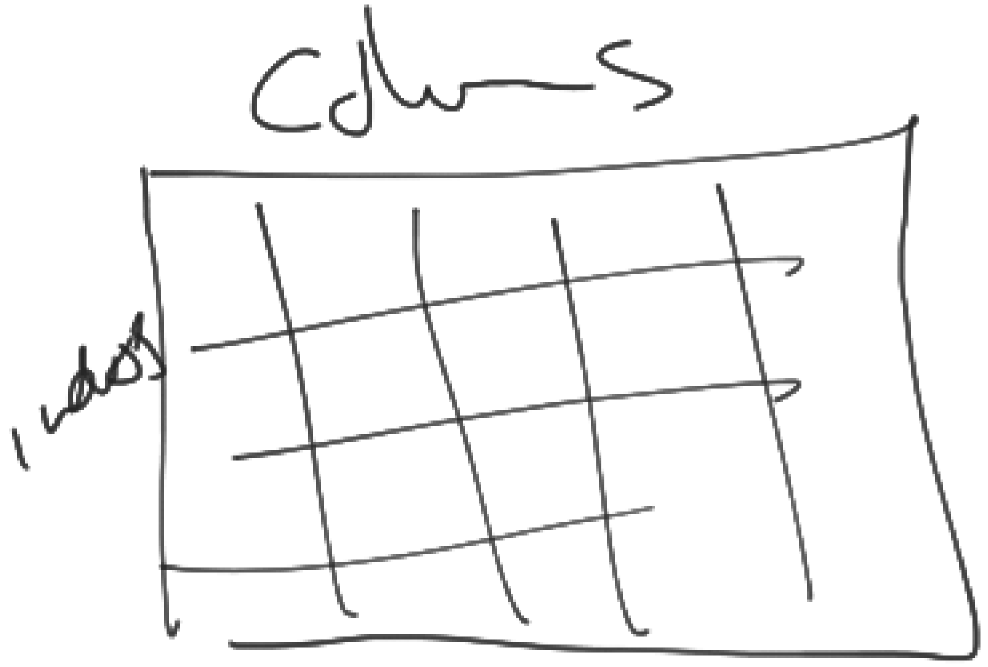
O(n) $O(n^2)$

O(nlogn)

Object Oriented Programming

Data + computation

- int
- str
- float
- dict
- list
- heap
- tuple



user-defined Data types

Objects

(a) has attributes
nouns variable

(b) can do methods
verbs function

2 Important

① Define object / class => instruction

② Create / instantiation => act

①

class Name :

attributes

methods

②

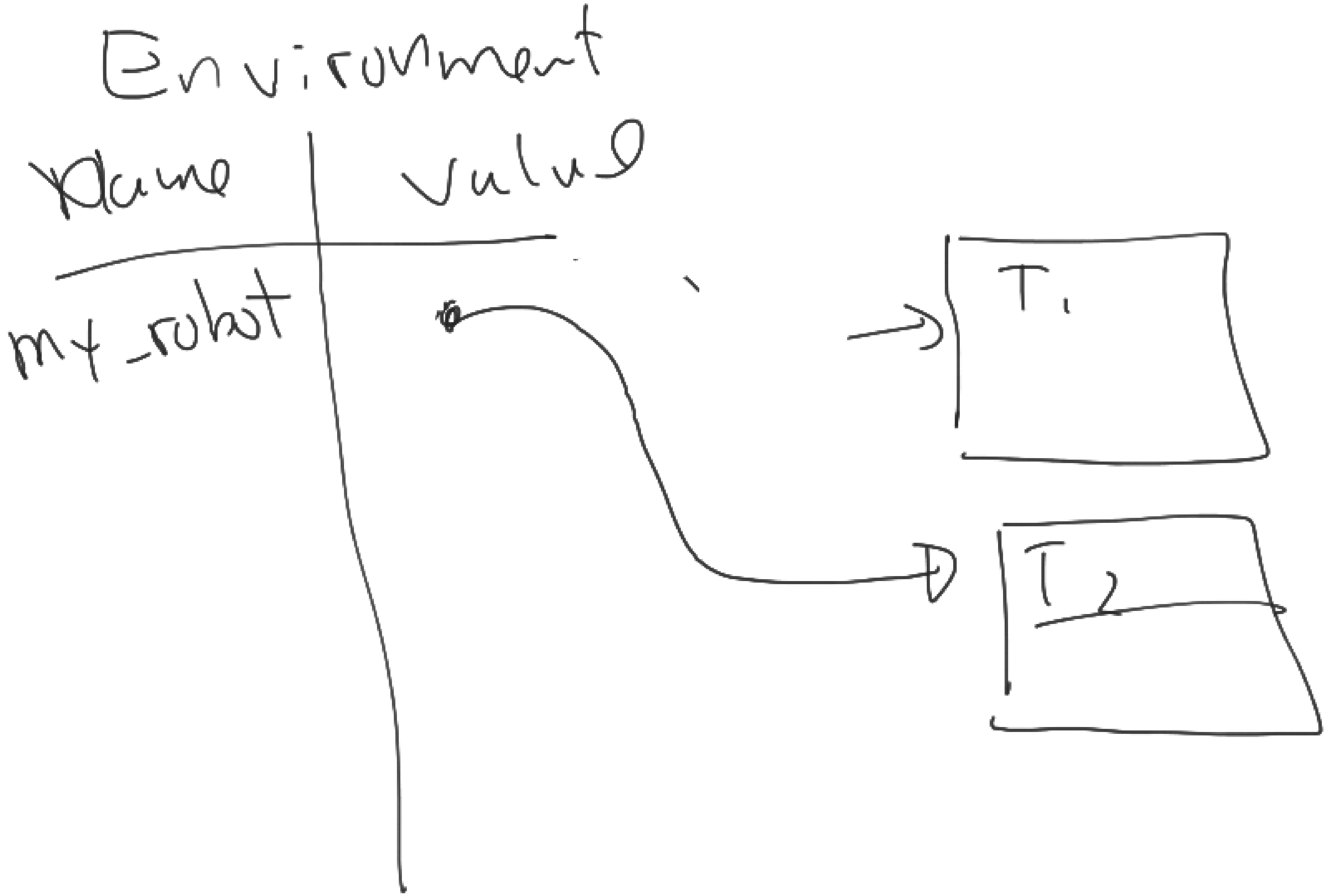
var = Name( )

initialize

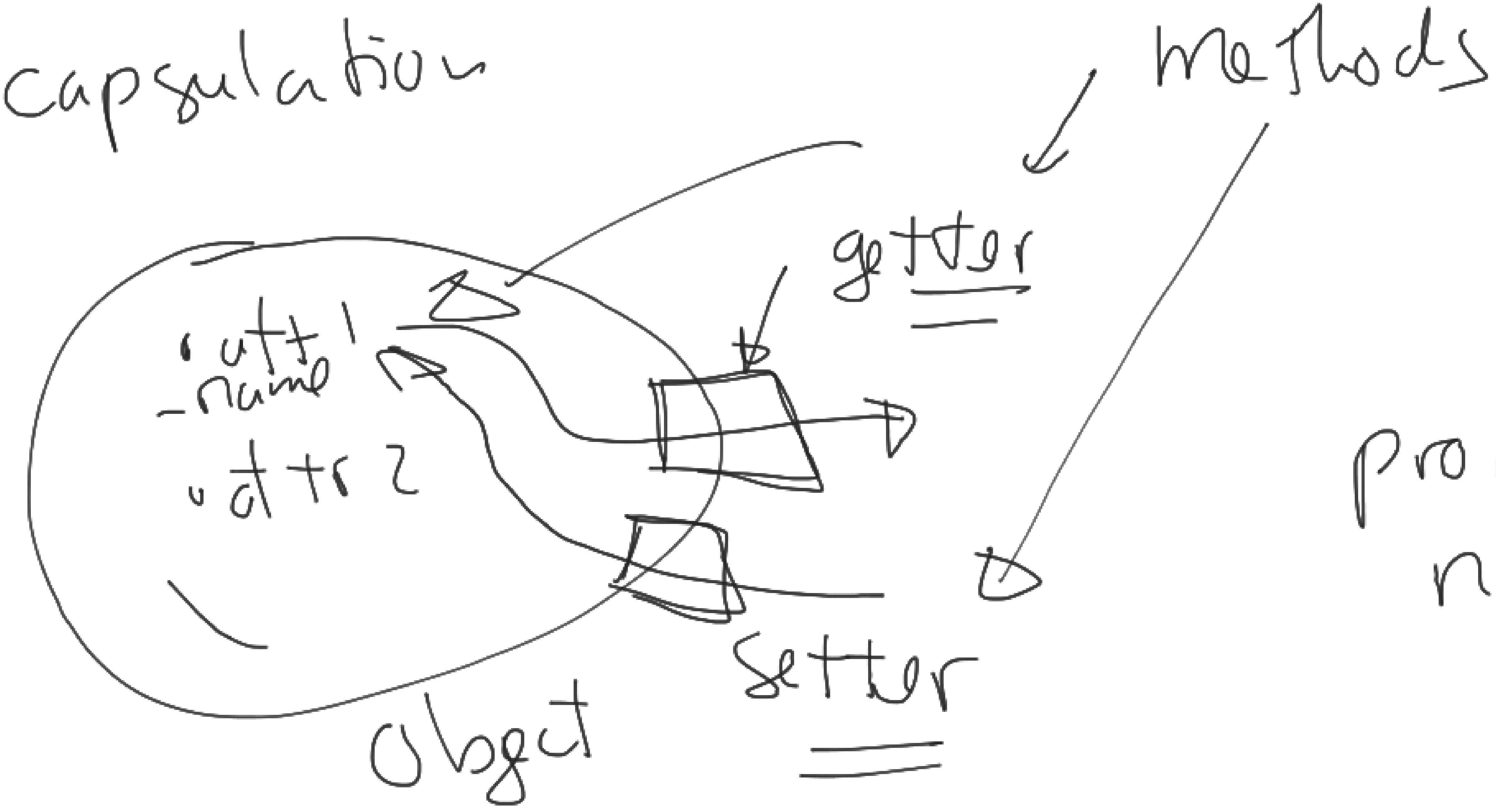
③ Dot operator

object . attribute

object . method( )


Environment
Name
value
my_robot
T_1
T_2

Encapsulation



property
name

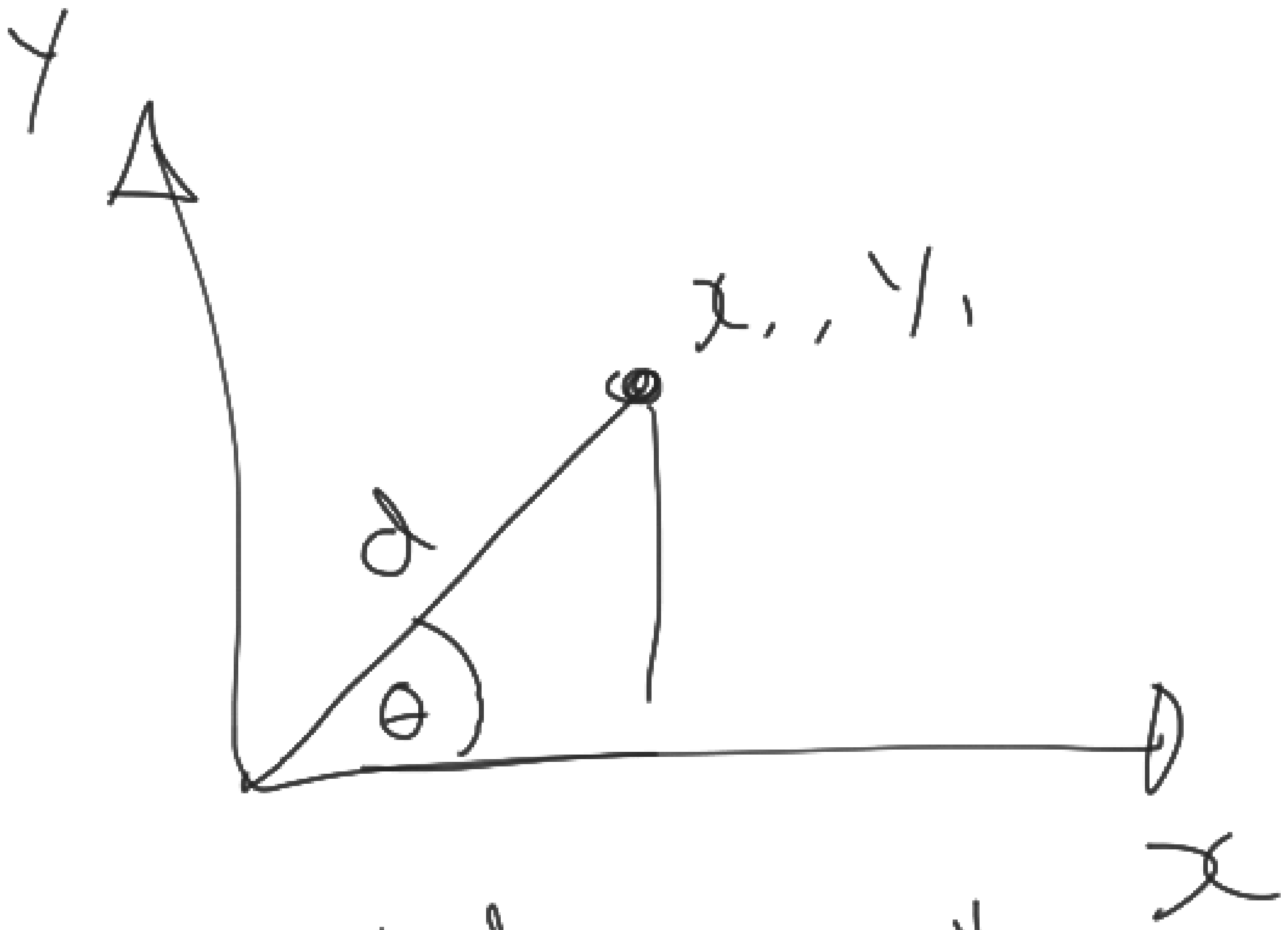
my_robot.name = "T3"

value

PEP8 convention

attribute		property
_name	→	name
_speed	→	speed

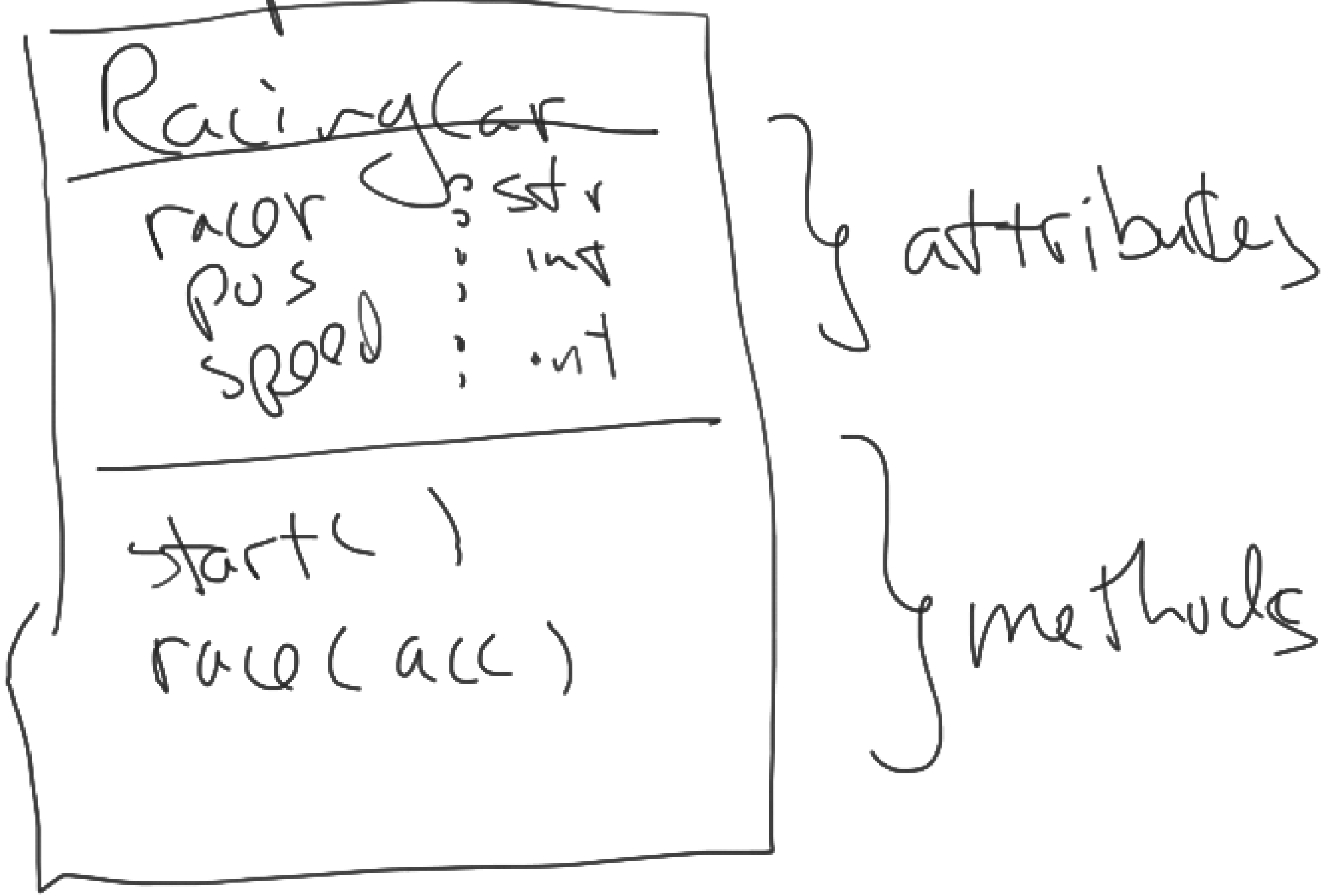
Coordinate



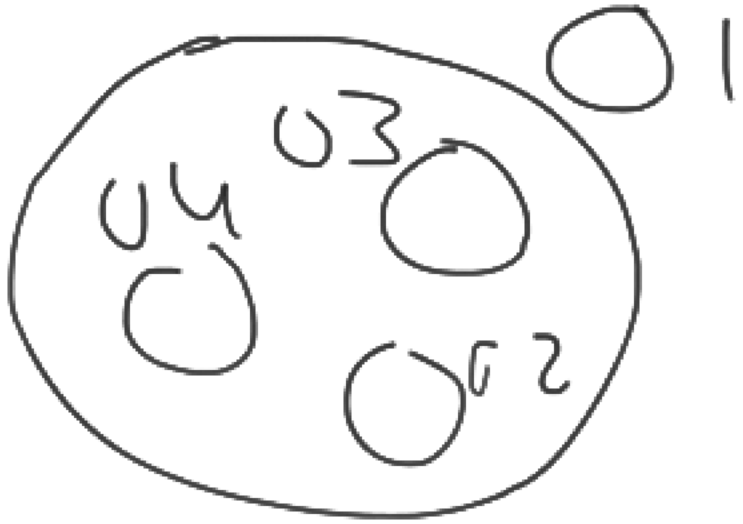
computed property

$$\text{distance} = \sqrt{x^2+y^2}$$

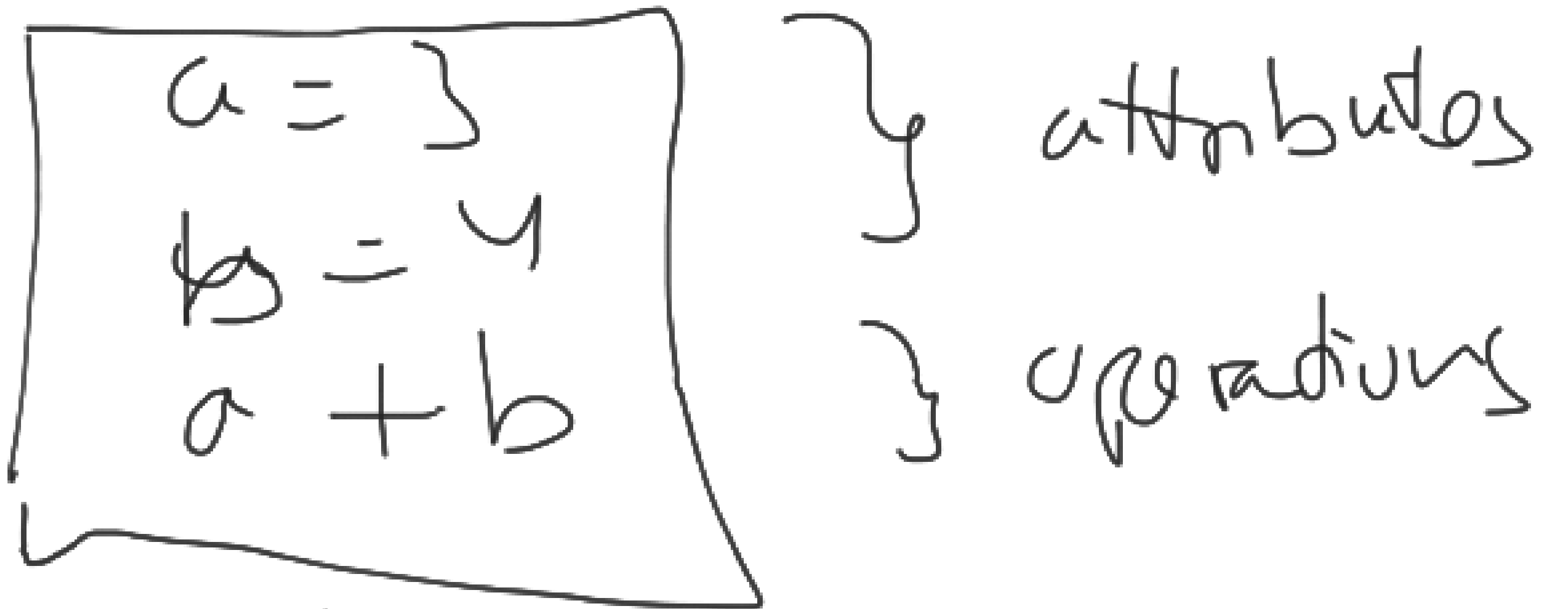
Composition



UML
class Diagram

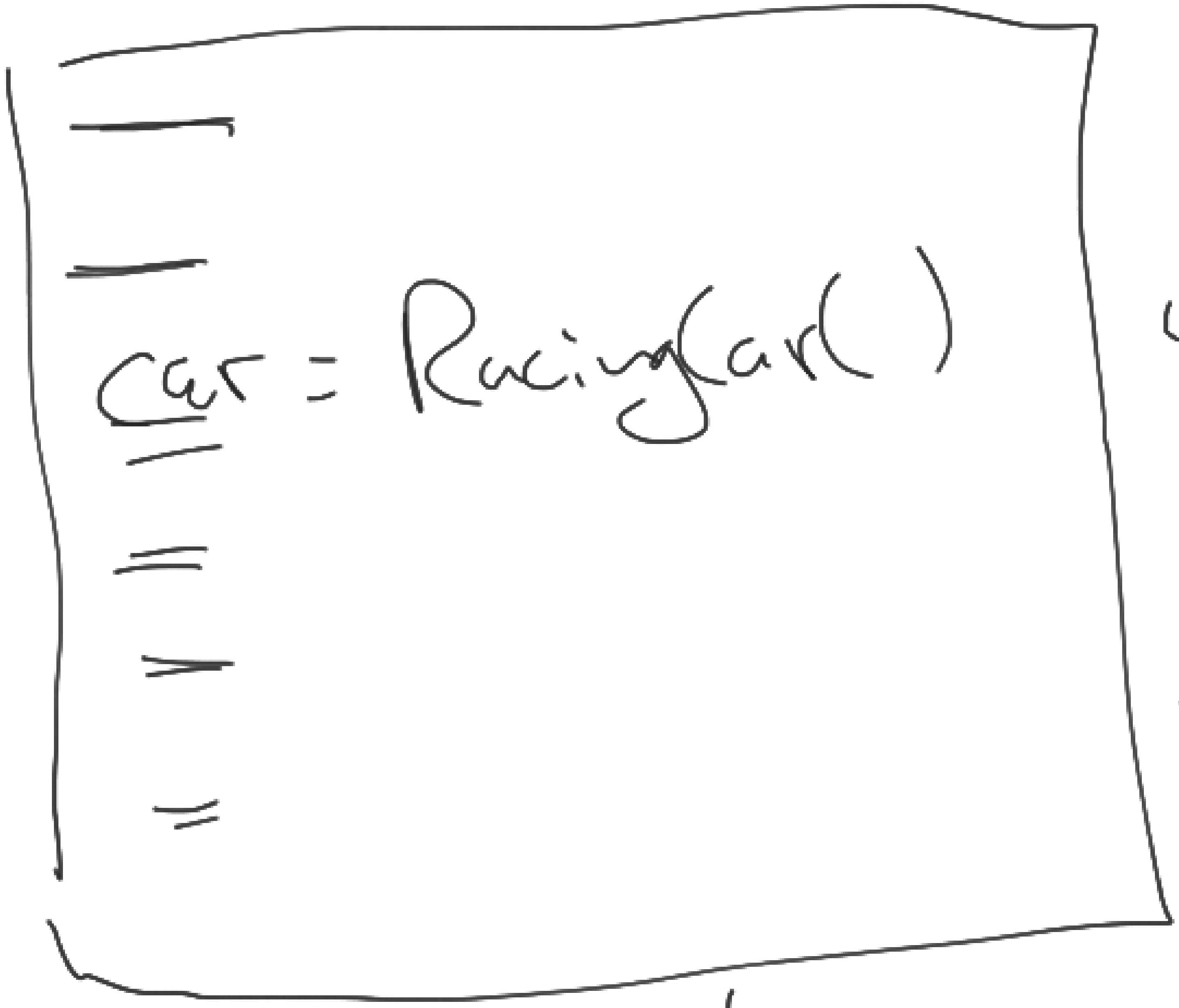


Abstraction

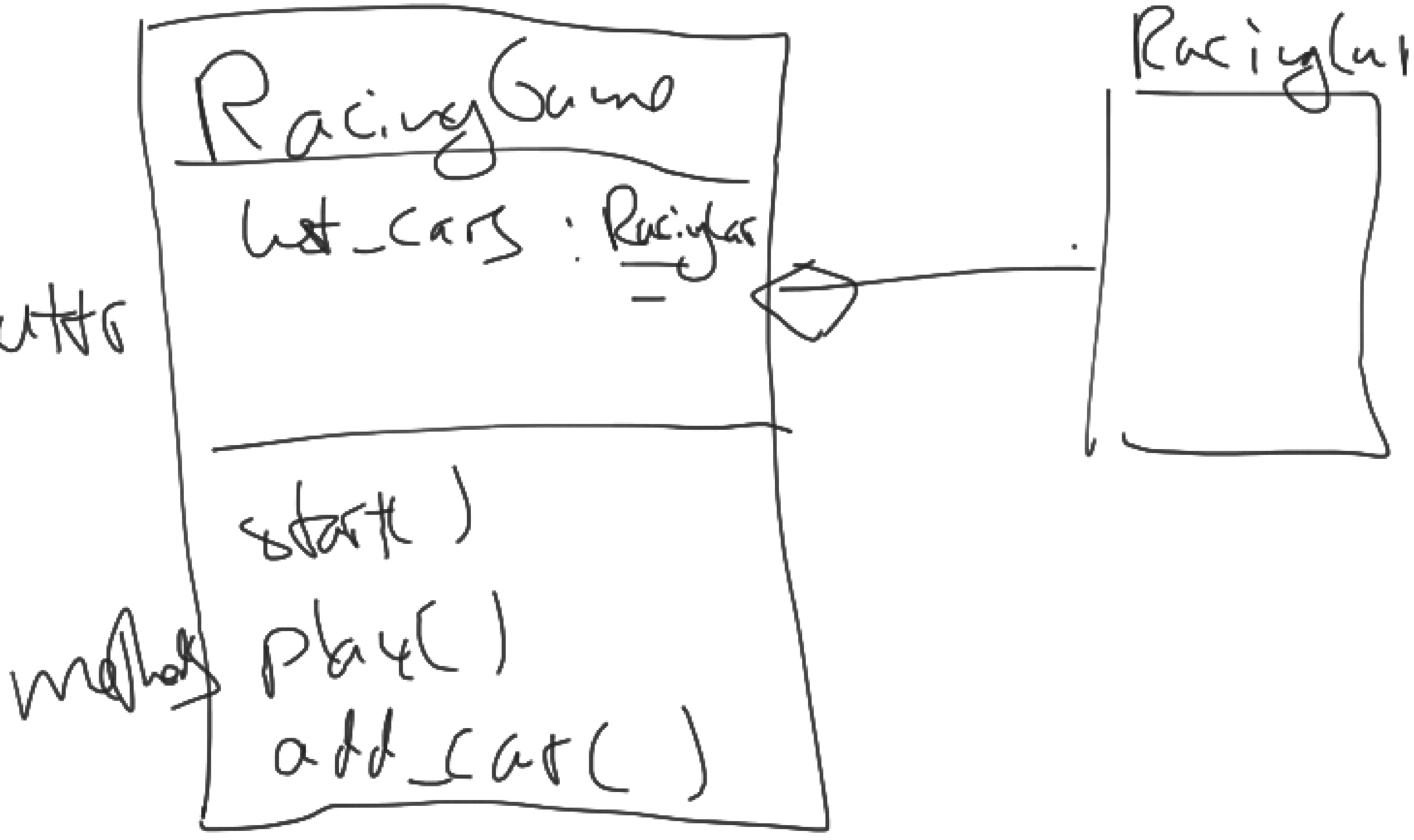


object

Code / program



object



Composition

Data Structures → OO

int

str

float

Avatar

Car

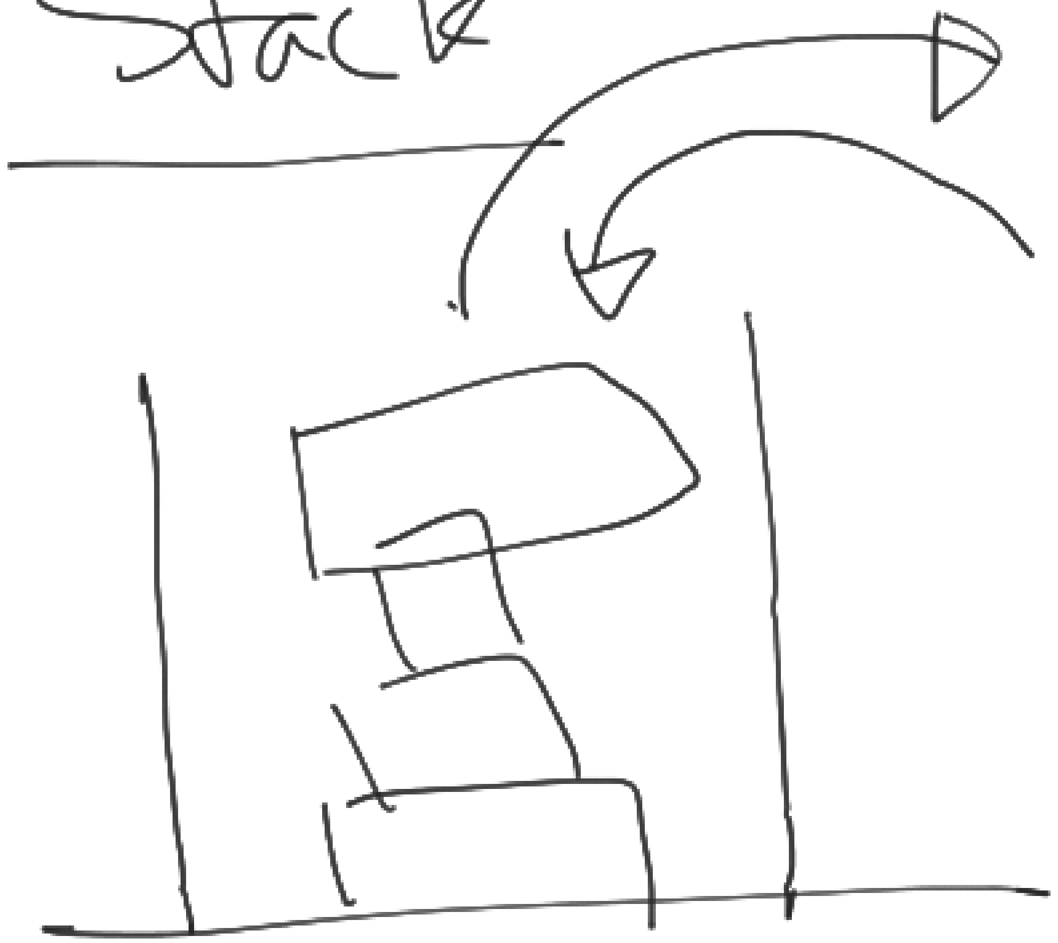
list → Abstract Data Type (ADT)

Linear → Stack

→ Queue

Graph

Stack



Stack	
#items list items	attributes

push(item)
pop( )
peek( )
	methods

$\underbrace{(4+3)} \times \underline{5}$ ← infix

4 + 3

operand operator

post-fix

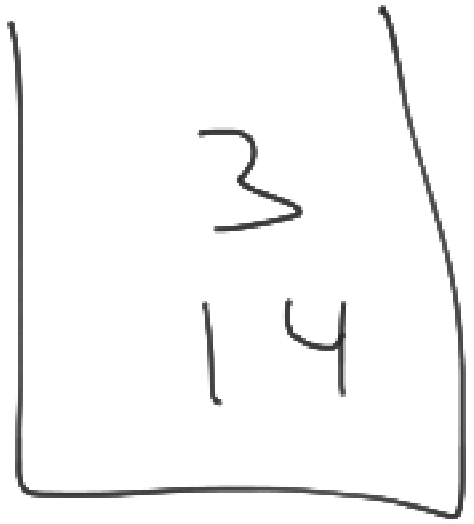
$4+3 \Rightarrow \underbrace{4\ 3\ +}$

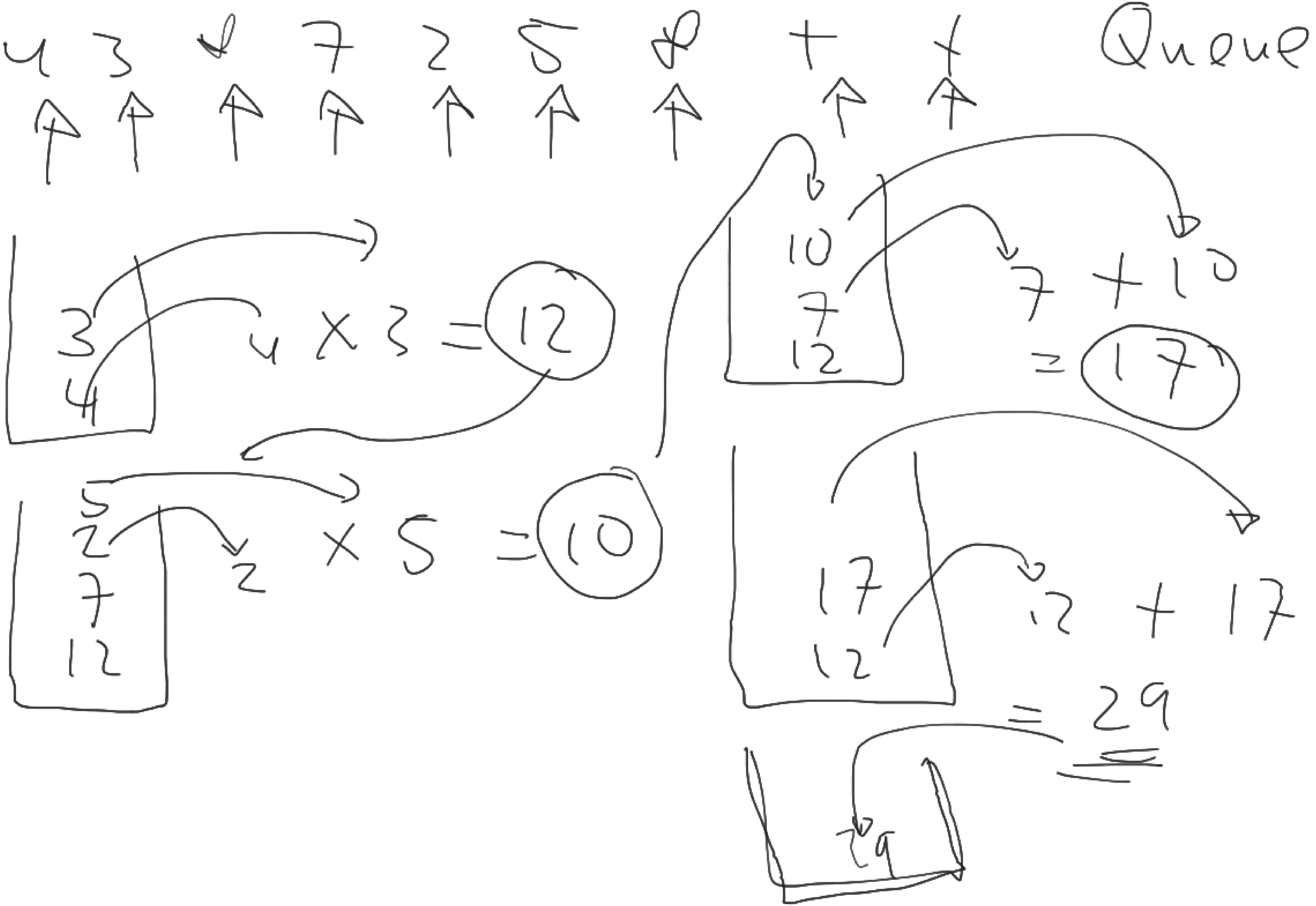
$(4+3)\times 5 \Rightarrow 4\ 3\ +\ 5\ \times$

$14+3$ $\quad 14\ \ 3\ \ +$

$4\ 3\ \times\ 7\ 2\ 5\ \times\ +\ +$

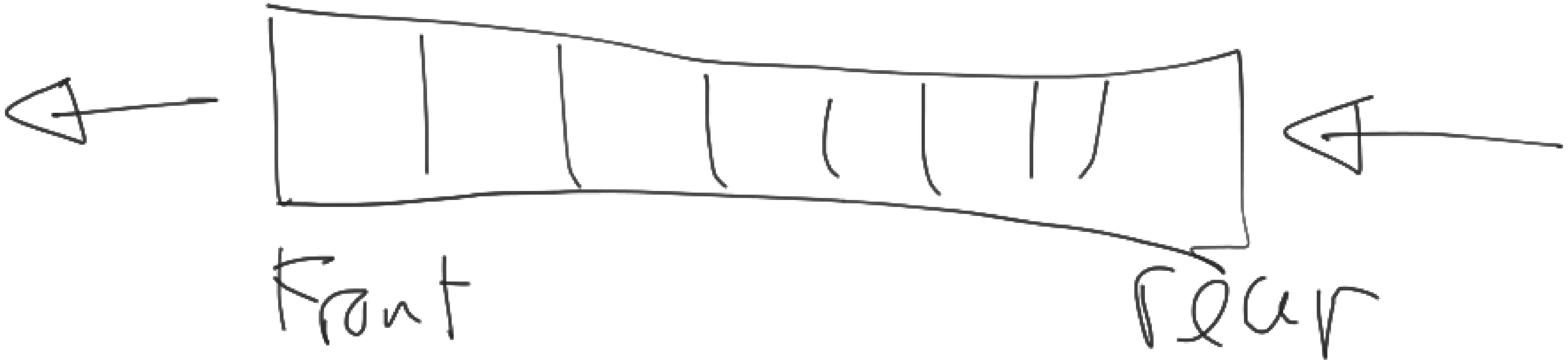
$(4\times 3)+((2\times 5)+7)$

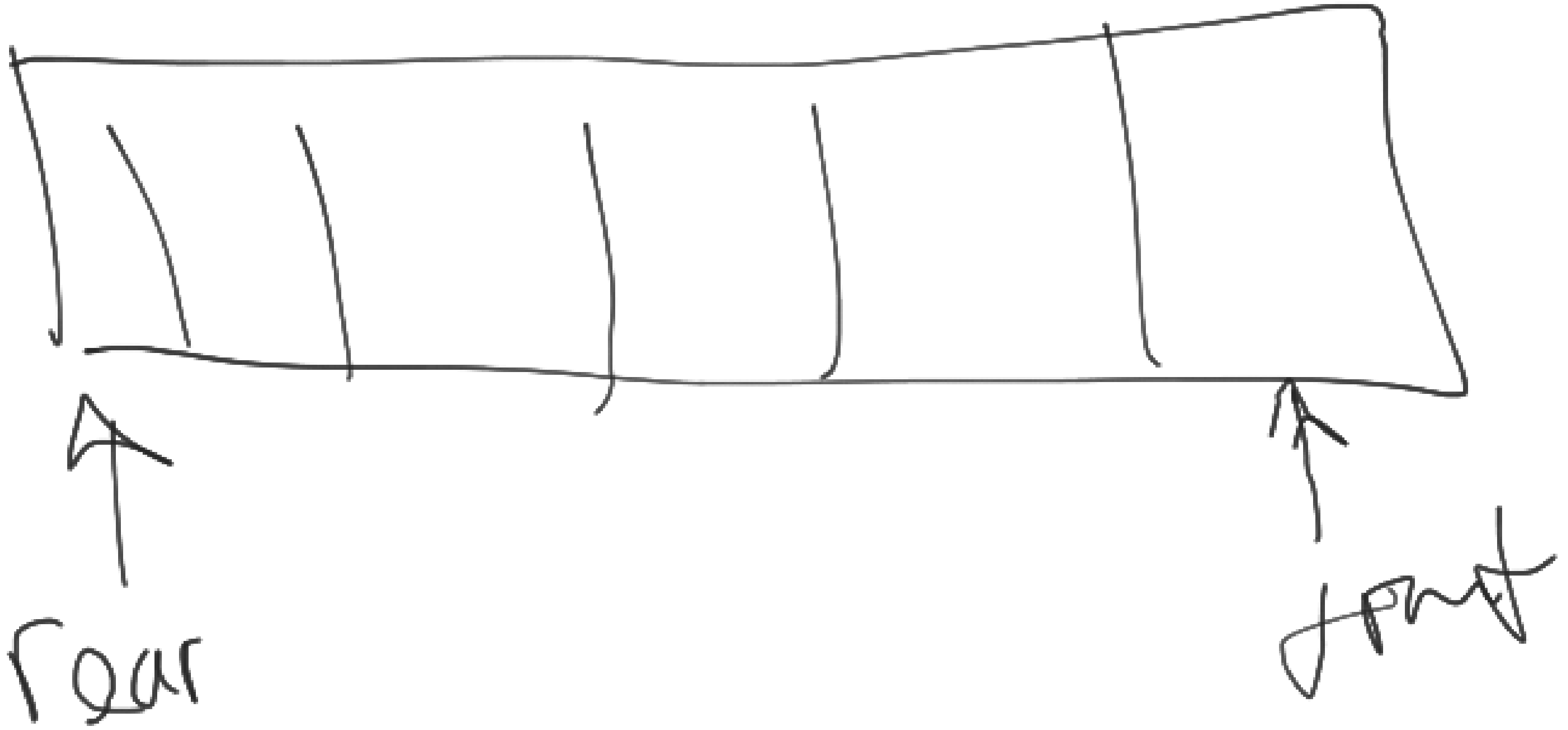



4 3 × 7 2 5 × + +
Queue
3
4
4 × 3 = 12
5
2
7
12
2 × 5 = 10
10
7
12
7 + 10
= 17
17
12
12 + 17
= 29
29

Queue

FIFO





pop( )

Design of Algo

```
1 As long as Queue is not empty
    1.1 Get item from Queue
    1.2 if item is not an operator
        1.2.1 Push item into stack
    1.3 Otherwise
        1.3.1 Pop op1 from stack
        1.3.2 pop op2 —"—
        1.3.3 evaluate
        1.3.4 Push result into stack
2 pop result
```

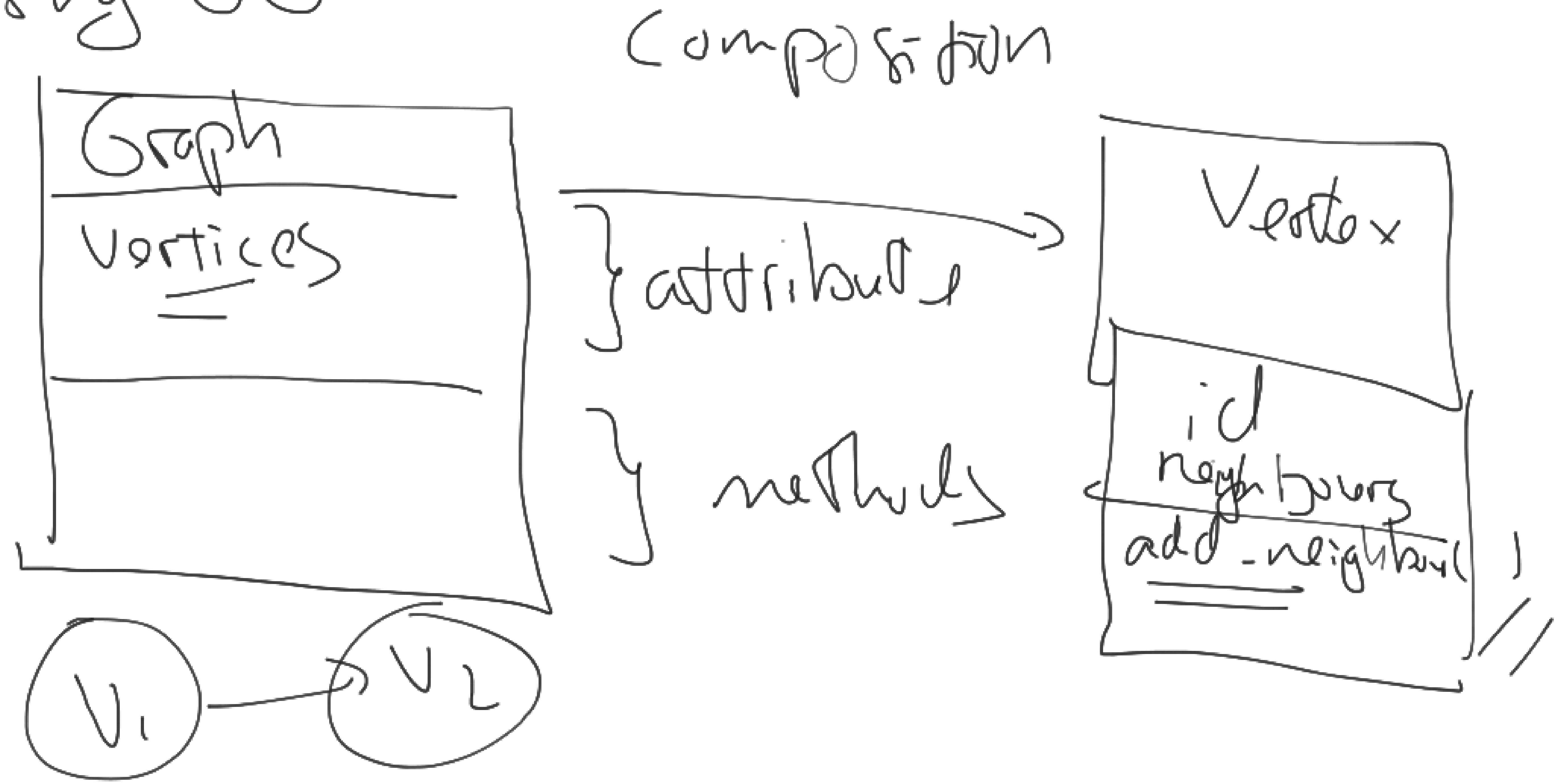
Graph using OO

Vertex

Edge

↓

implicit



{ 'V1' : V1 '

'V1' : V2 '

"has-a" → composition

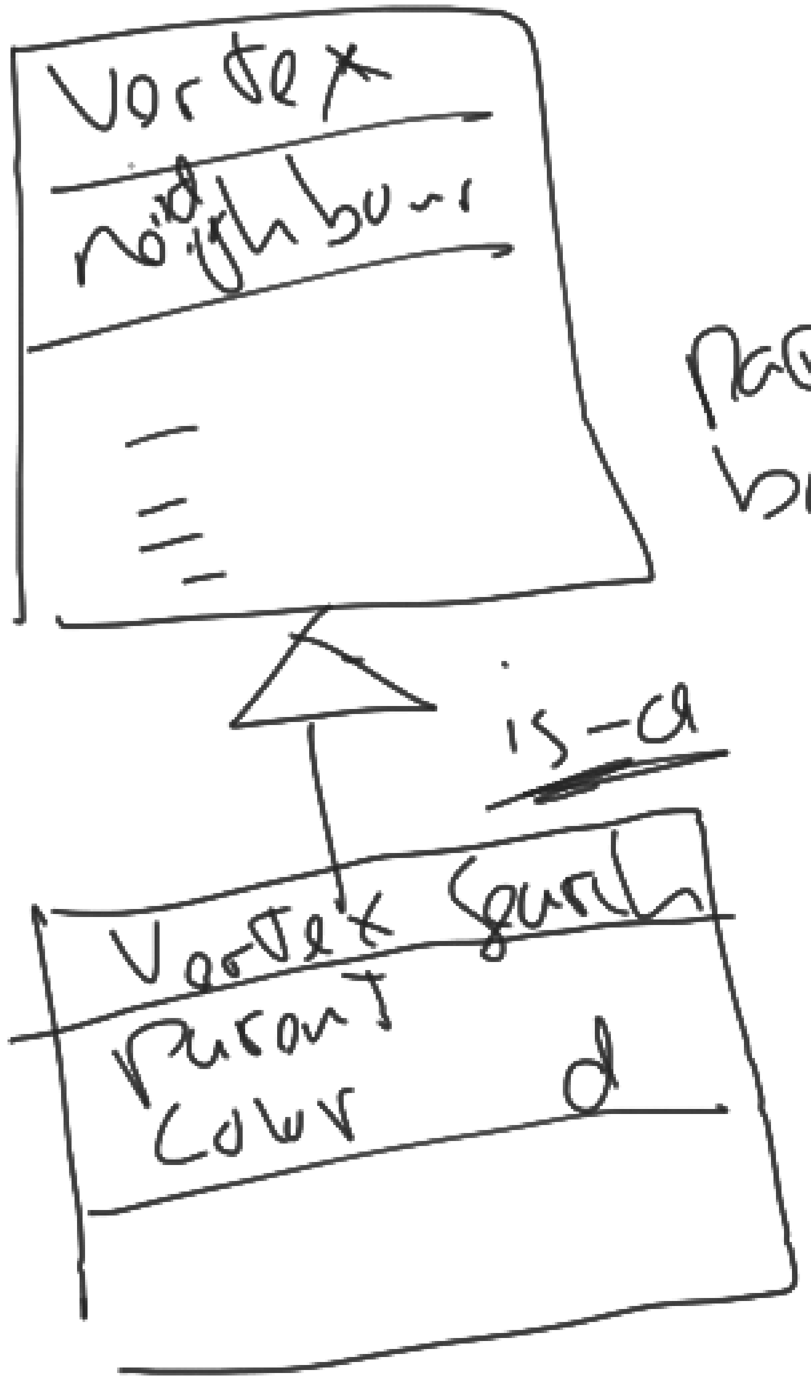
"is-a" → inheritance

1 Open/close principle

↓ extension ↓ modification

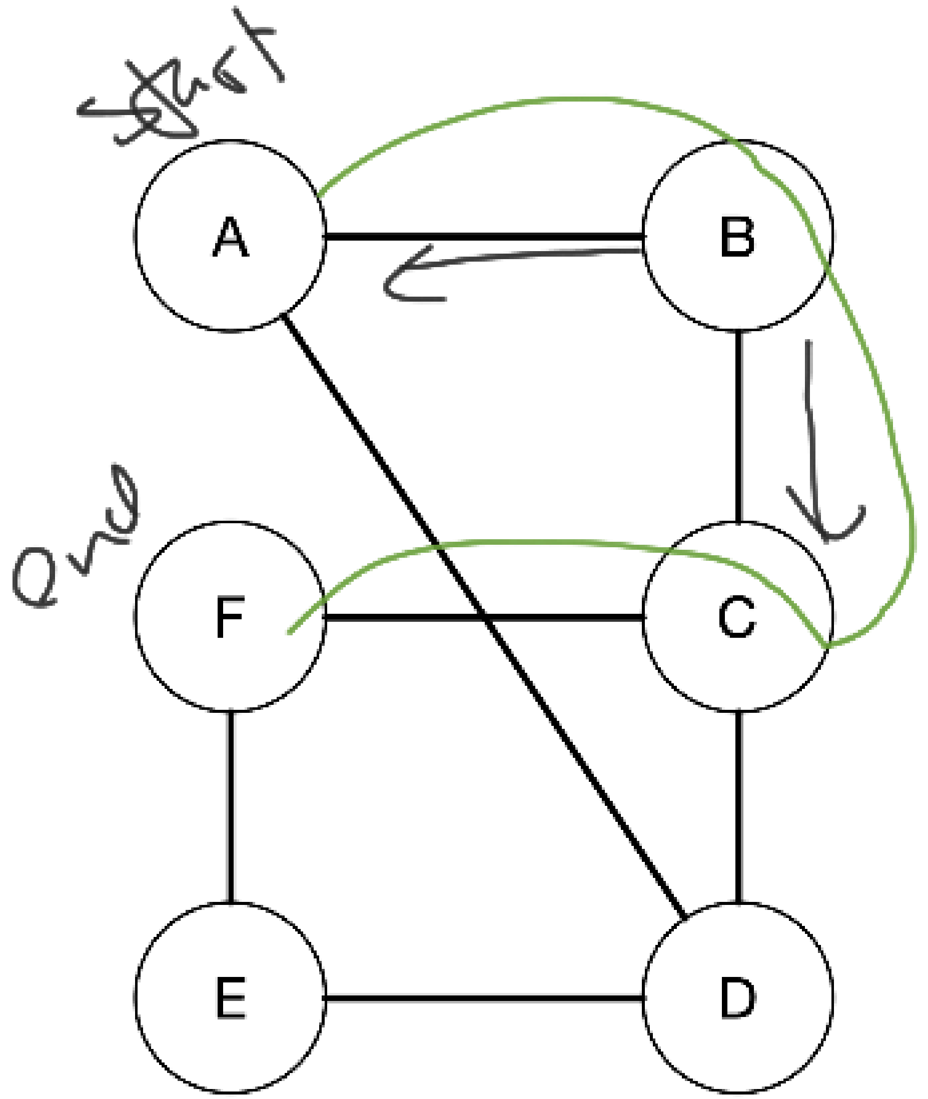
2 Liskov substitution principle

3 DRY



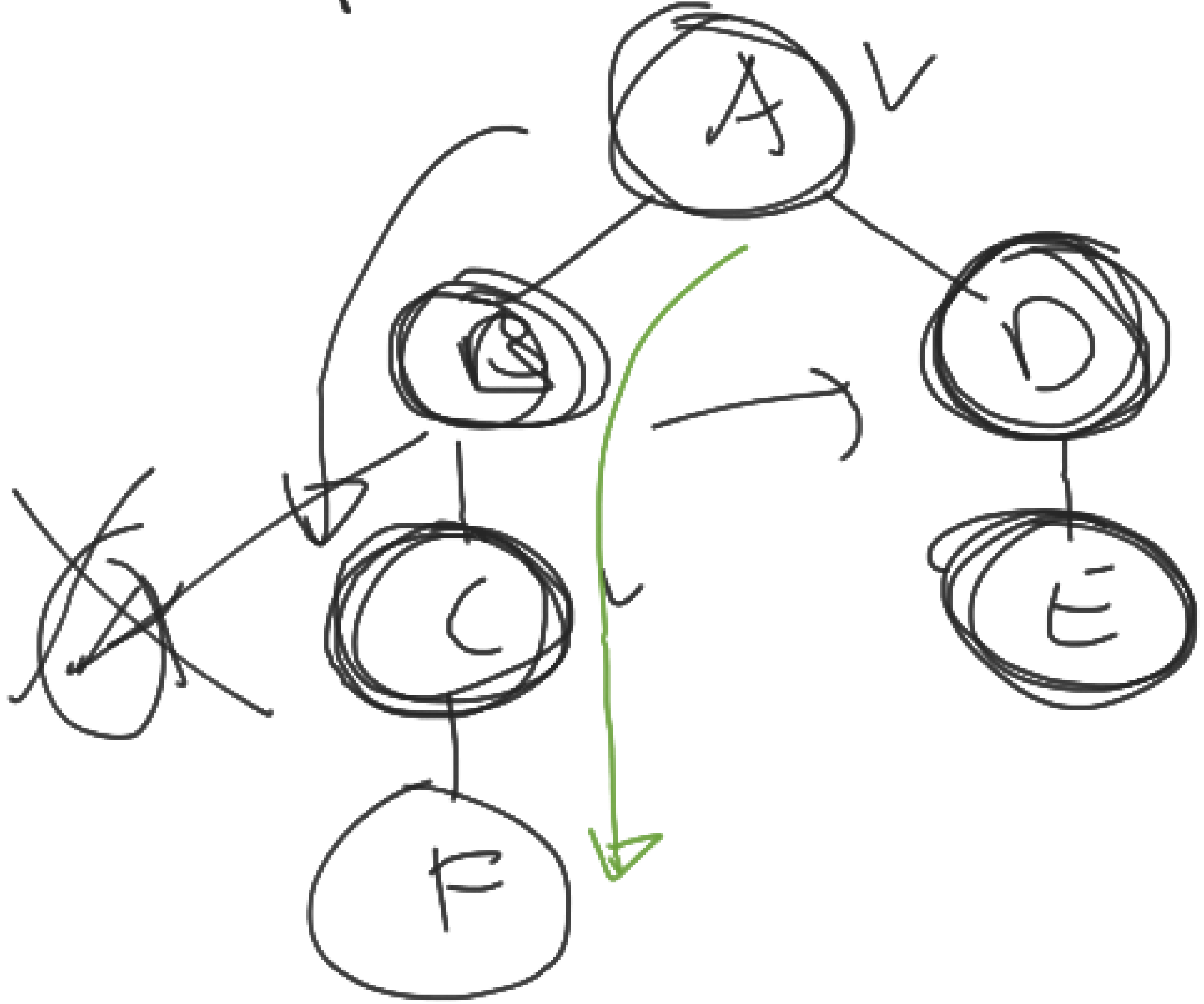
parent
base

child
subclass

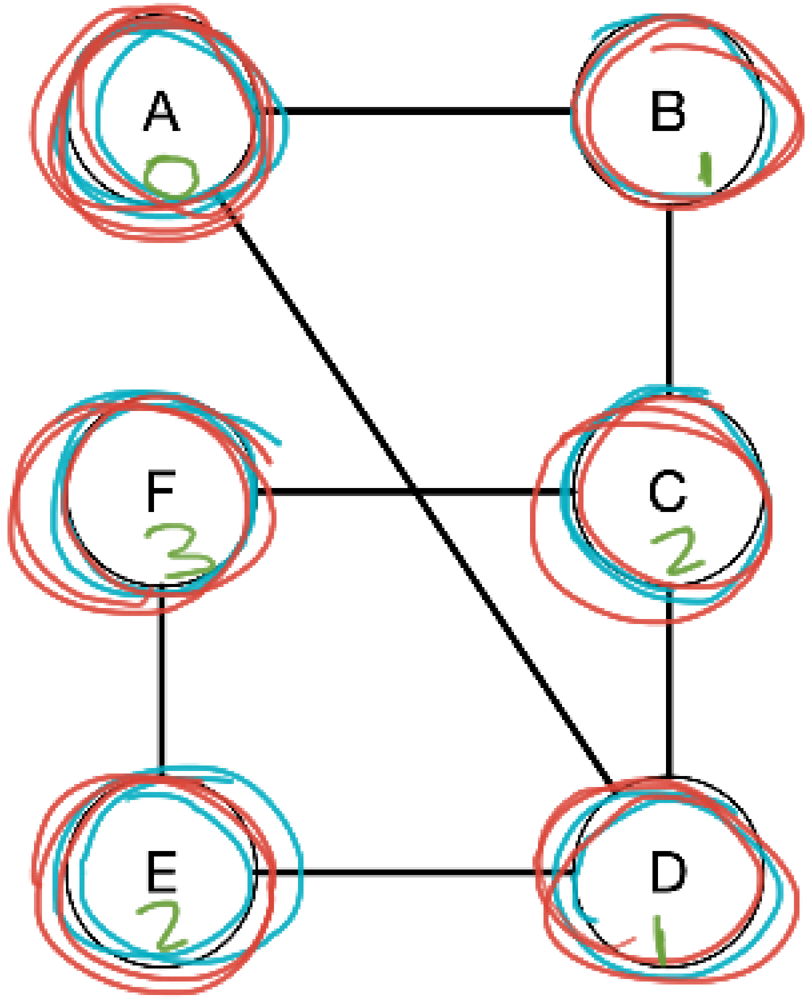


Search → Breadth First
↳ Depth First

Maps → shortest path



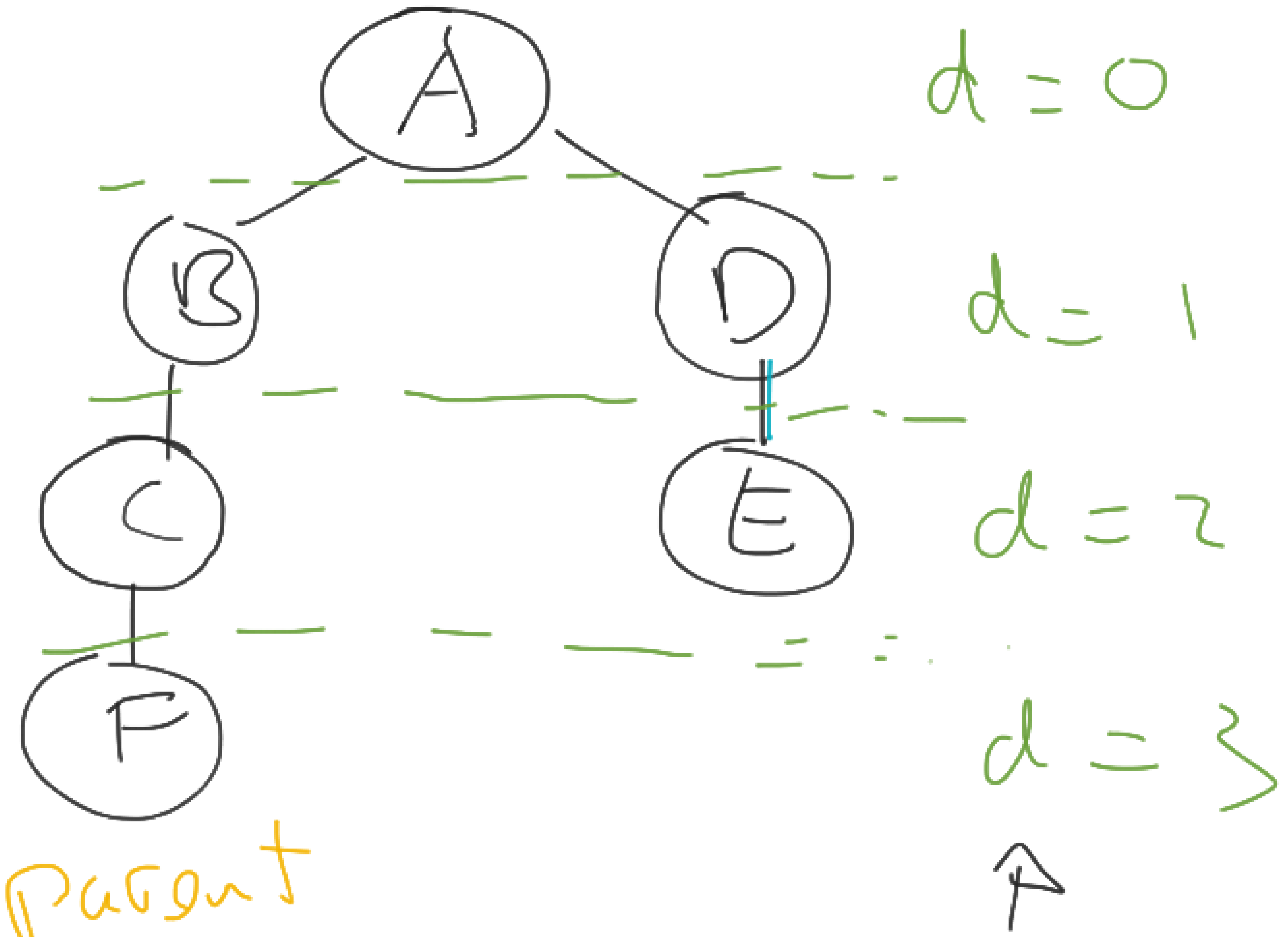
mark (a) visited
(b) explored
all the neighbours



grey → visited

black → explored all neighbours

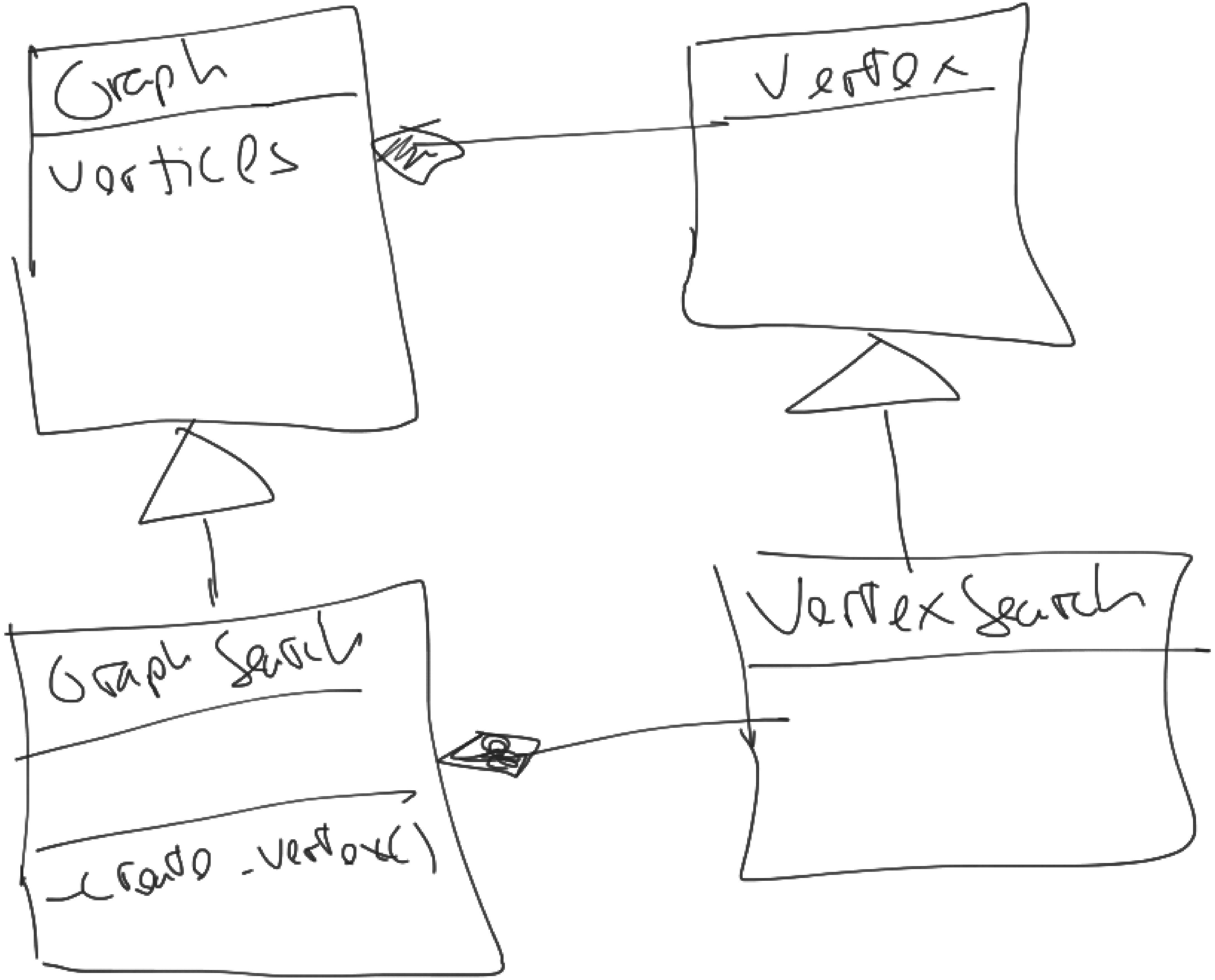
color

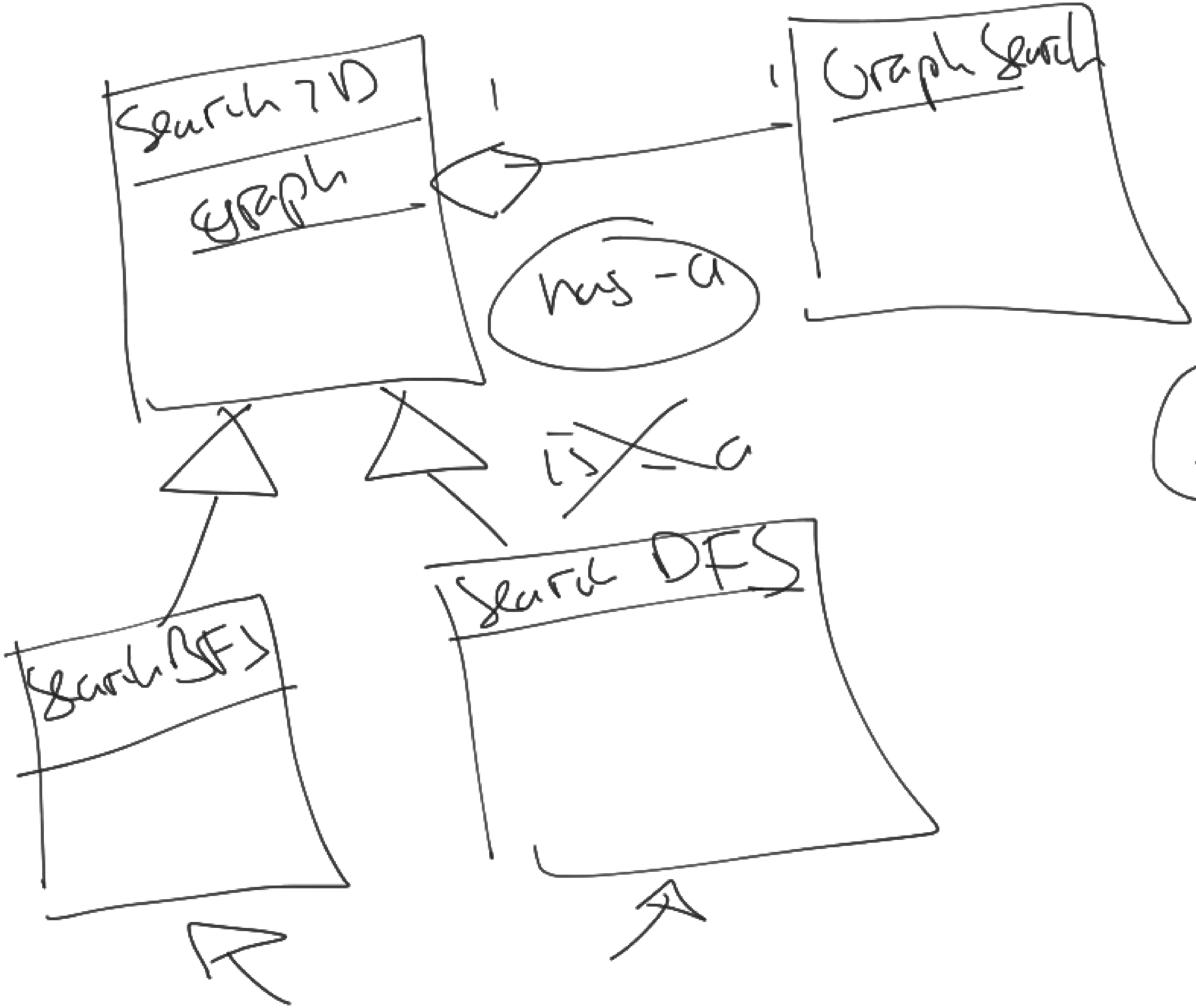


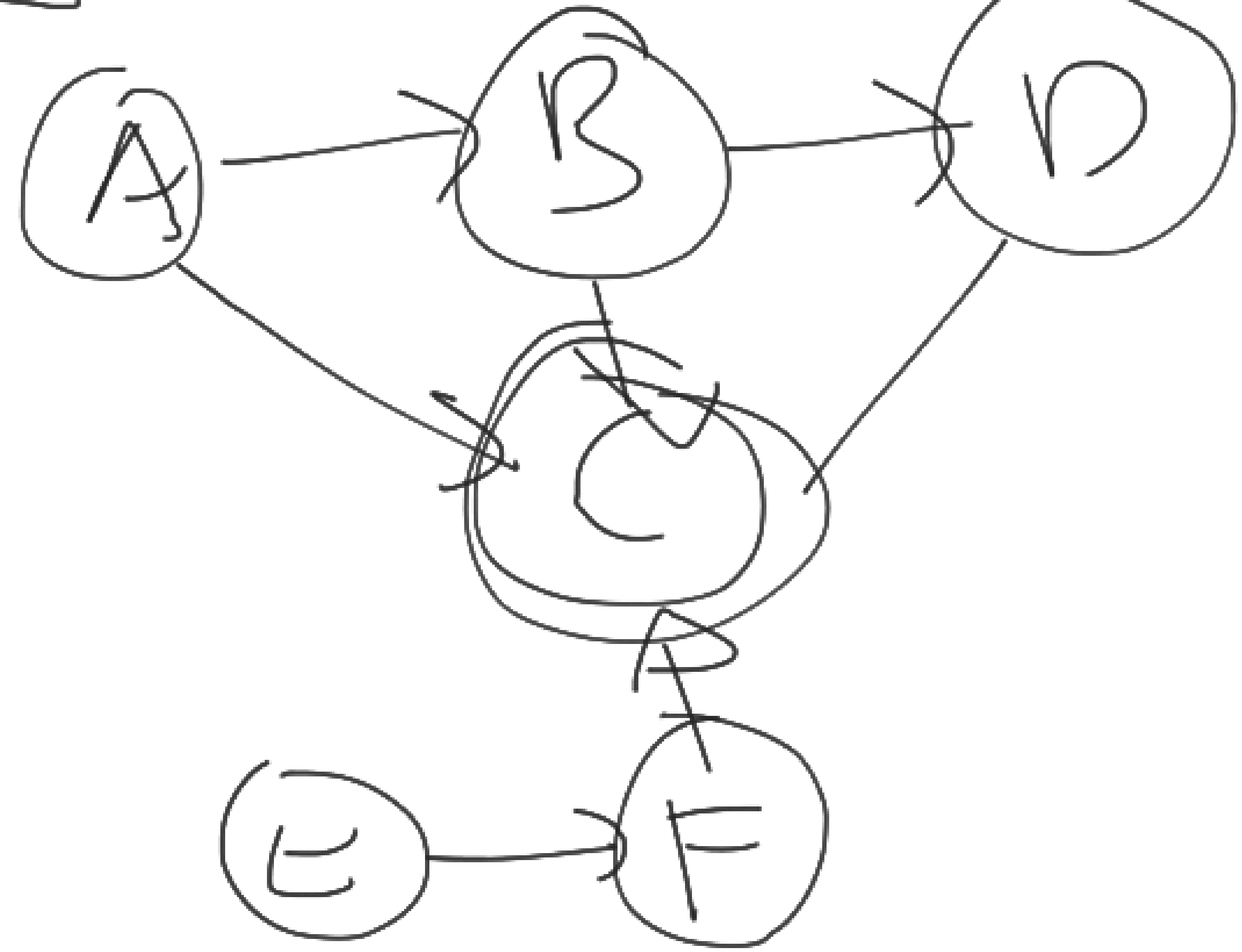
Parent

Front

Rear


Graph
vertices
Vertex
GraphSearch
_create_vertex()
VertexSearch


Search7D
Graph
1
1
GraphSearch
has-a
is-a
SearchBFS
Search DFS


A
B
D
C
E
F