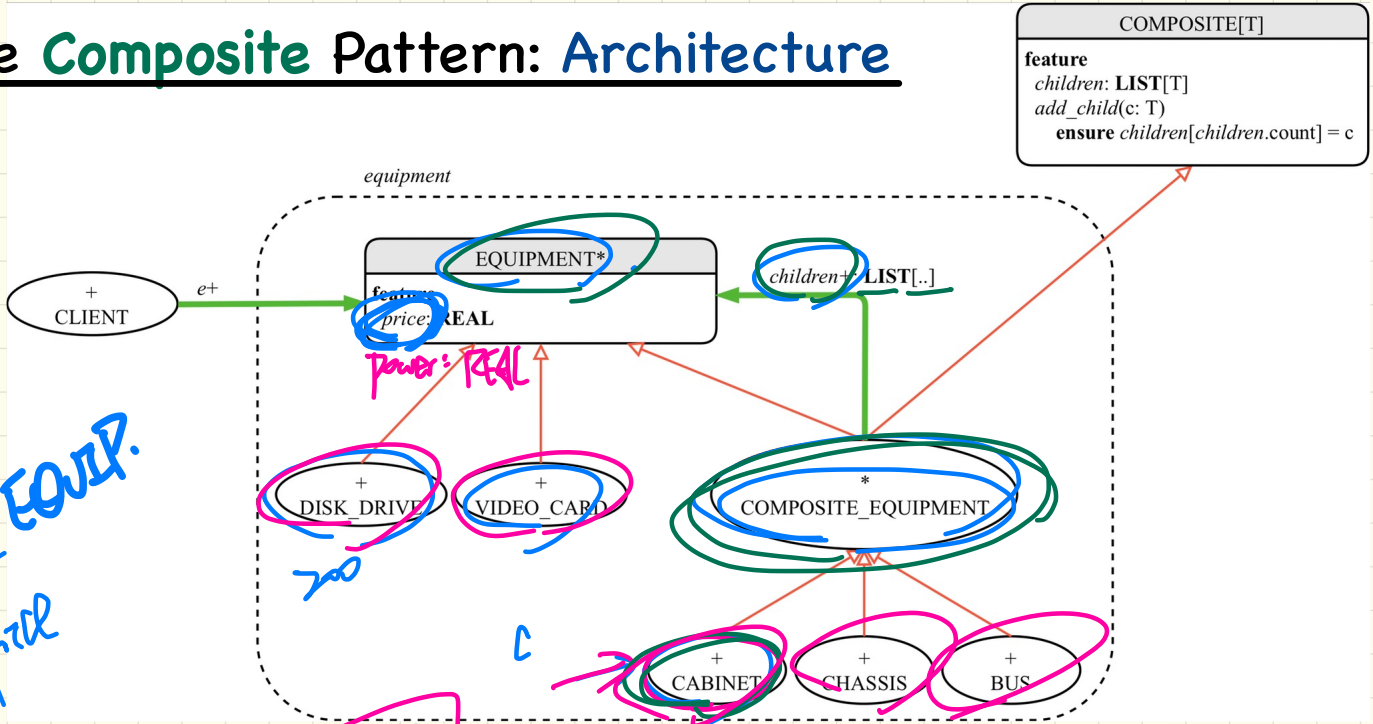
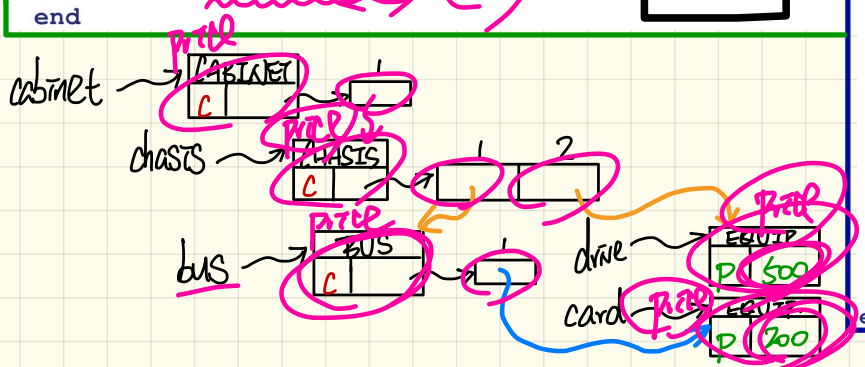
Lecture 21
Monday March 23

The Composite Pattern: Architecture



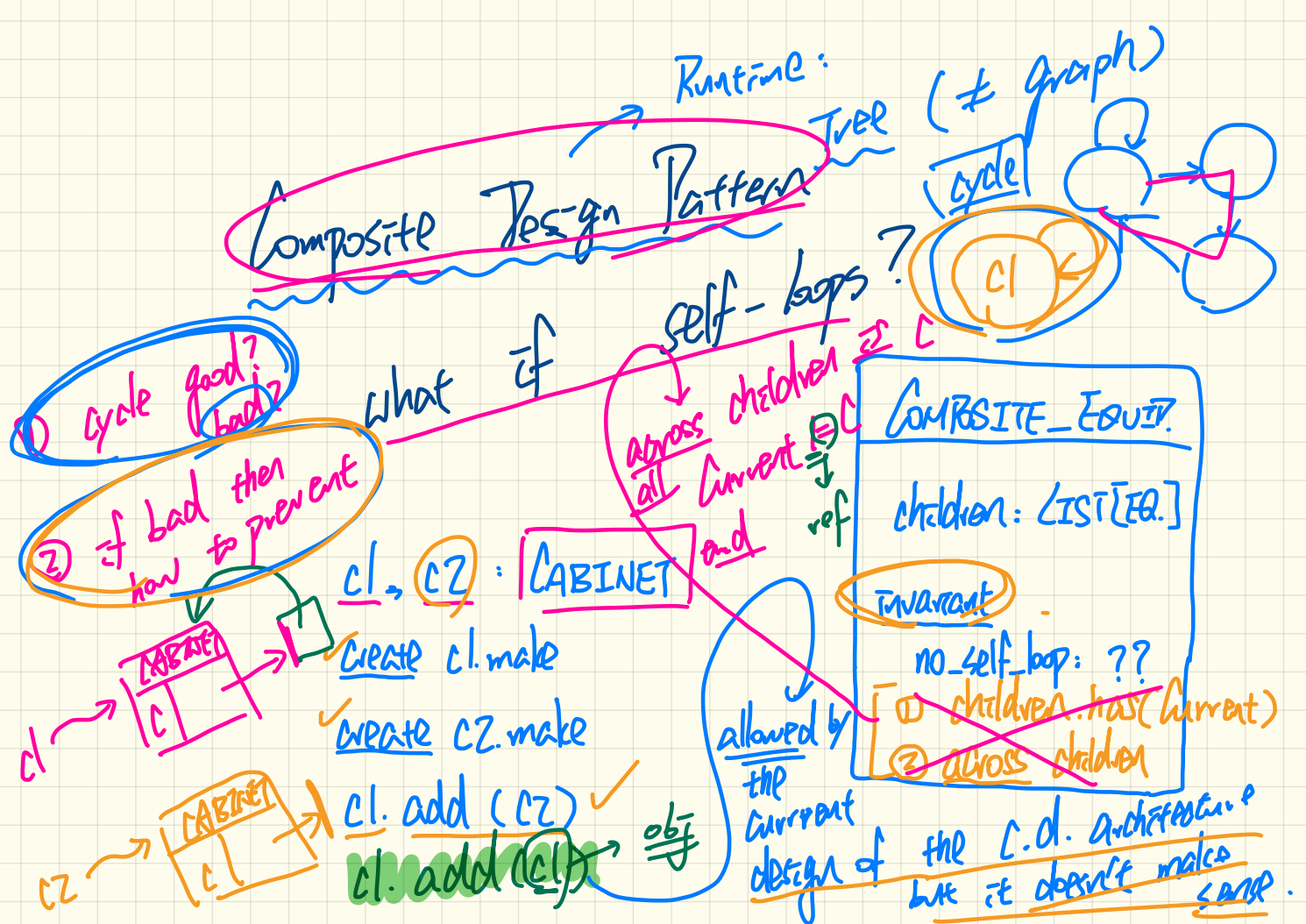
Testing the Composite Pattern

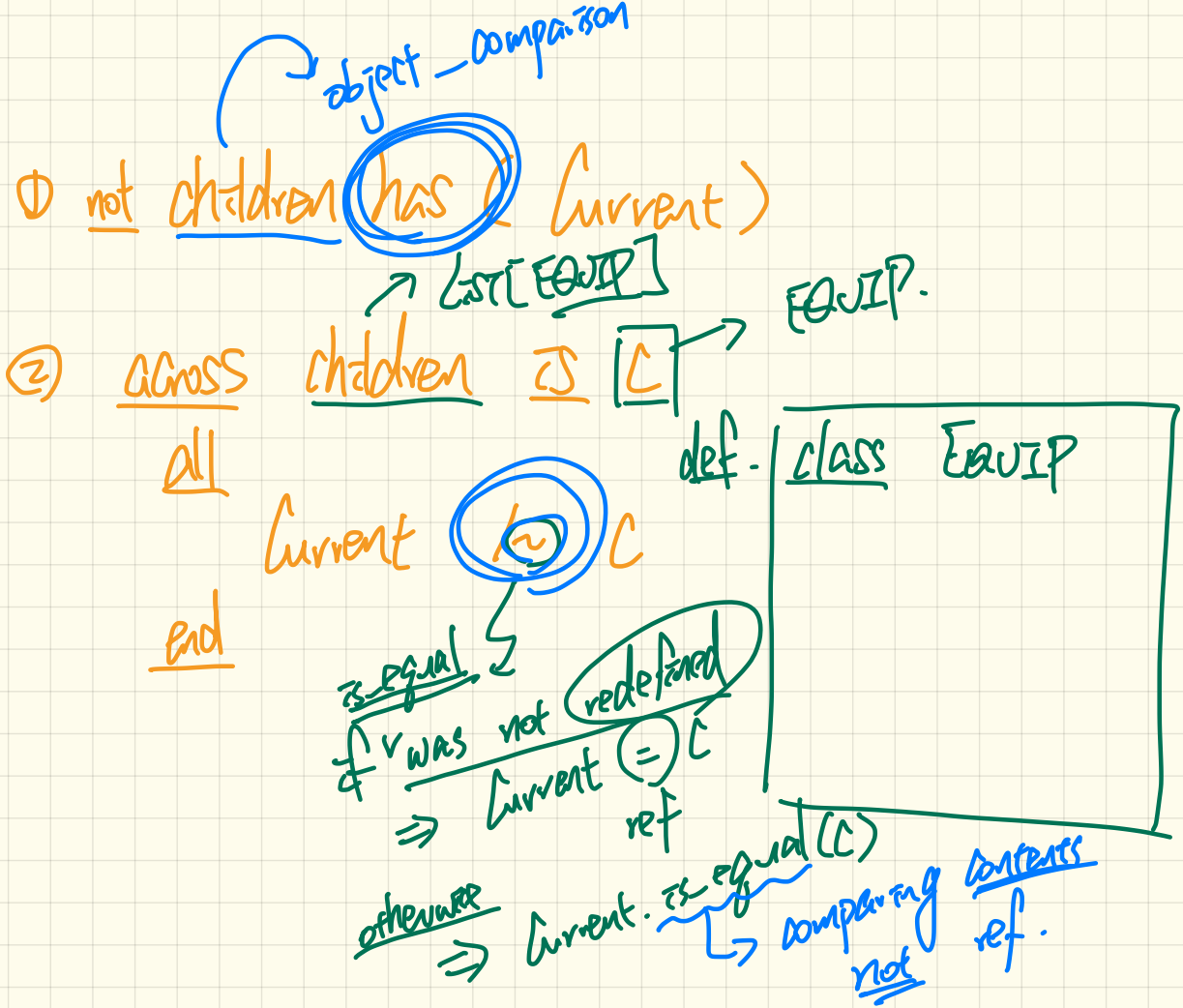
```
test_composite_equipment: BOOLEAN
  local
    card, drive: EQUIPMENT
    cabinet: CABINET -- holds a CHASSIS
    chassis: CHASSIS -- contains a BUS and a DISK_DRIVE
    bus: BUS -- holds a CARD
  do
    create {CARD} card.make("16Mbs Token Ring", 200)
    create {DISK_DRIVE} drive.make("500 GB harddrive", 500)
    create bus.make("MCA Bus")
    create chassis.make("PC Chassis")
    create cabinet.make("PC Cabinet")

    bus.add(card)
    chassis.add(bus)
    chassis.add(drive)
    cabinet.add(chassis)
    Result := cabinet.price = 700
  end
```



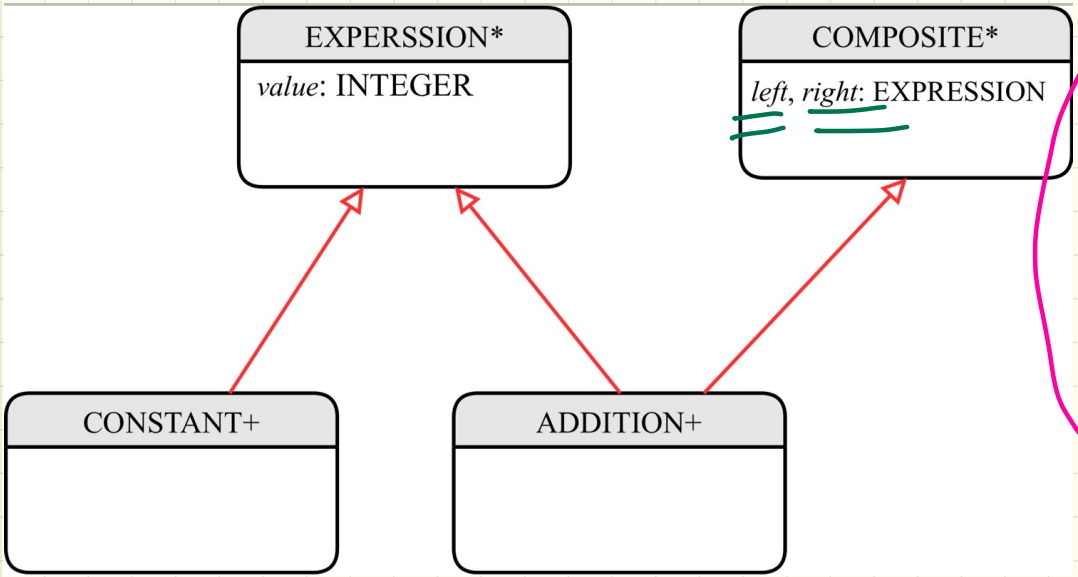

```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is
    end
end
```

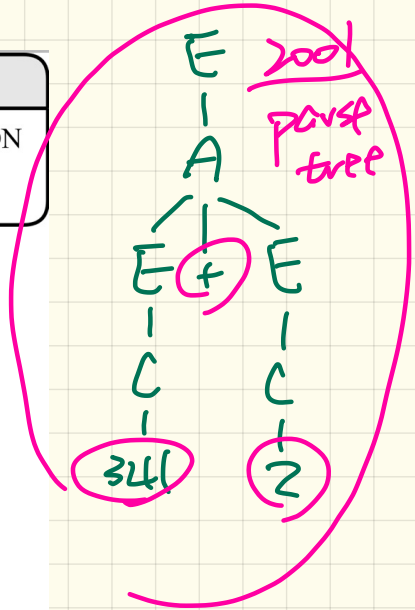
```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
      -- Sum the net prices of all sub-equip
    do
      across
        children as cursor
      loop
        Result := Result + cursor.item.price
      end
    end
end
```


Runtime:
Composite Design Pattern: Tree (≠ graph)
cycle
C1
what if self-loops?
① cycle good? bad?
② if bad then how to prevent
across children ⇒ C
across all current ⇒ C
ref
COMPOSITE_EQUIP.
children: LIST[EQ.]
invariant
no_self_loop: ??
① children.has(current)
② across children
C1, C2 : CABINET
create C1.make
create C2.make
C1.add(C2) ✓
C1.add(C1) obj
allowed by the current design of the C.D. architecture but it doesn't make sense.


object_comparison
① not children.has (Current)
LIST[EQUIP]
EQUIP.
② across children is C
all
Current /~ C
end
def. class EQUIP
is_equal
if v was not redefined
Current = C
⇒ Current ref
otherwise
⇒ Current.is_equal(C)
comparing contents
not ref.

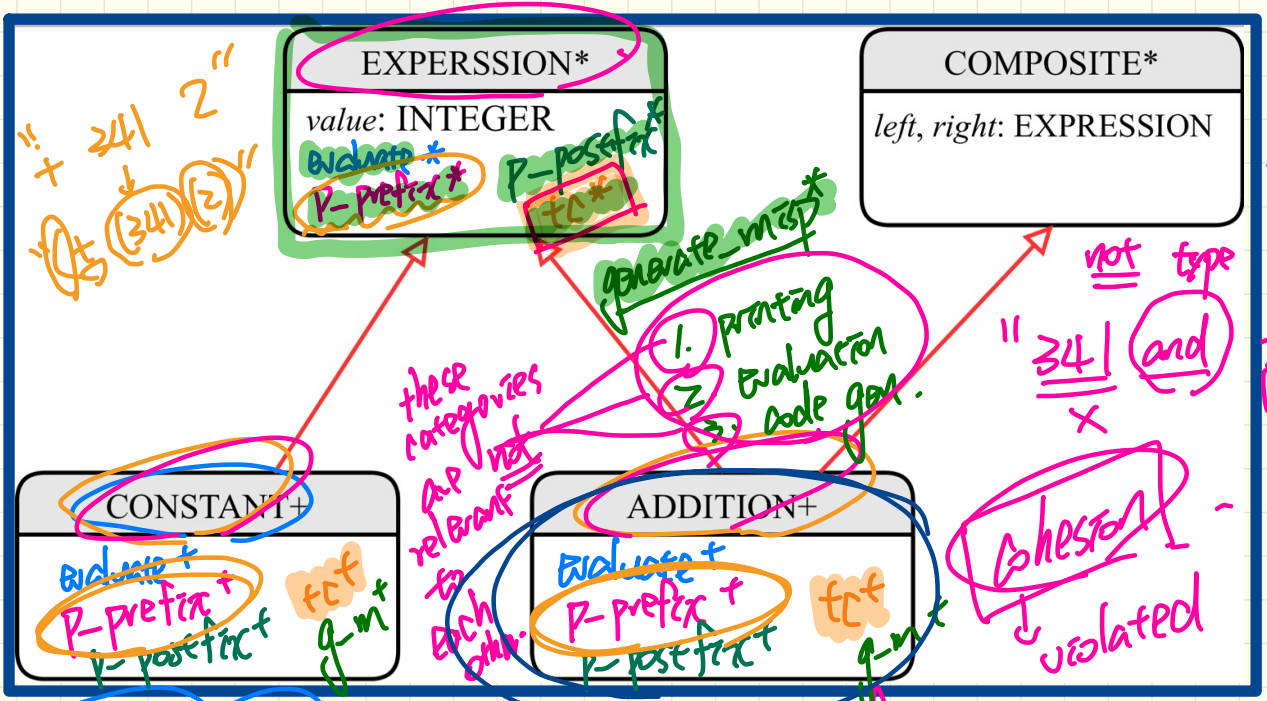
Design of Language Structure: Composite Pattern





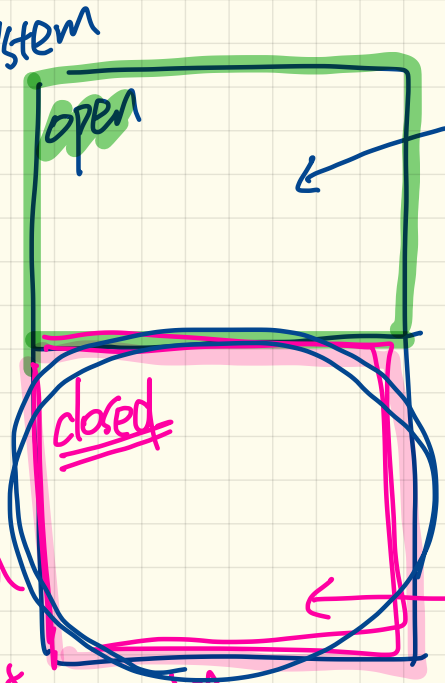
Q: How do you construct a composite object representing "341 + 2"?

Design of Language Operation: How to Extend the Composite Pattern?



Open/closed Principle

do we satisfy OCP?



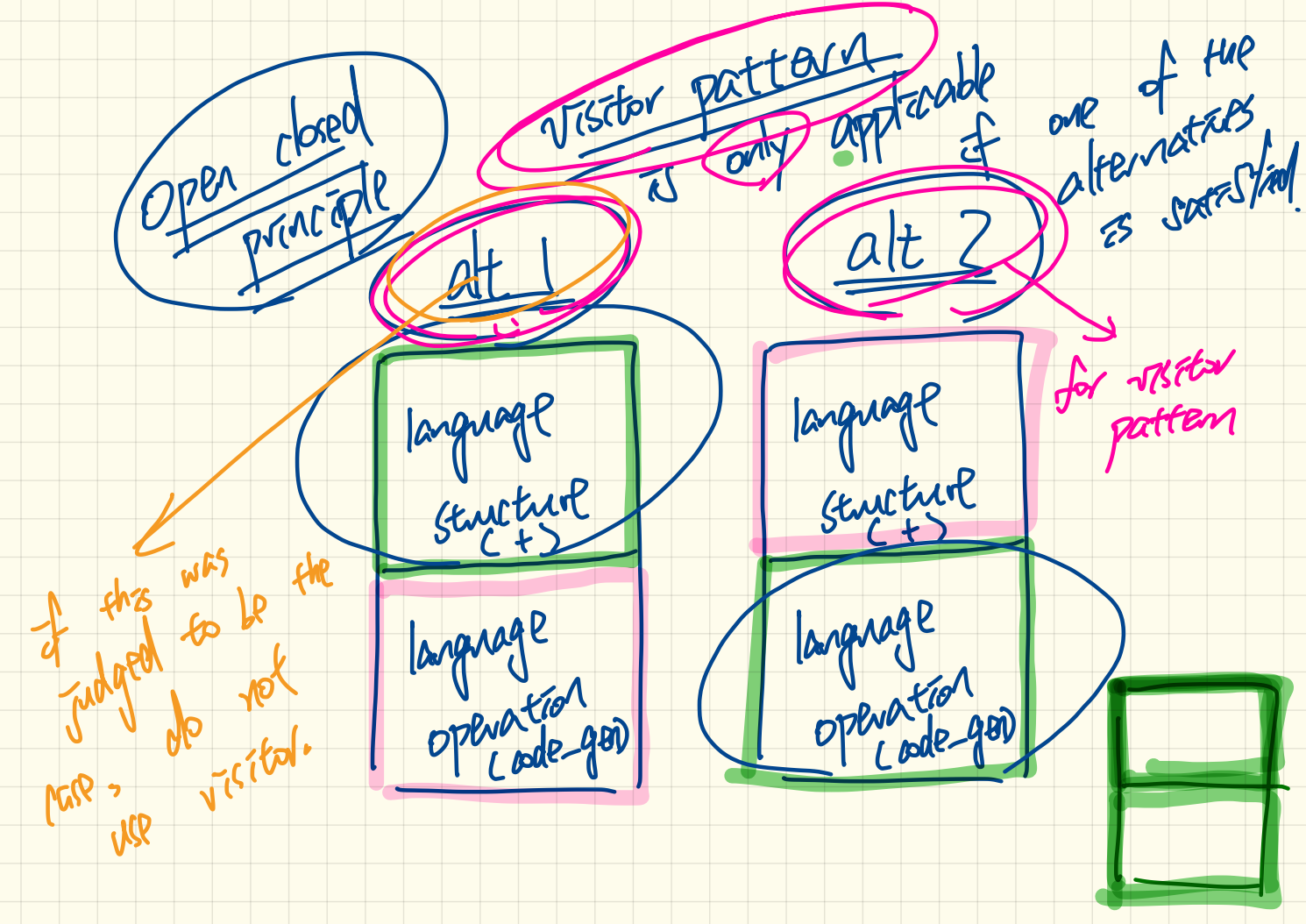
↓

Yes if there's a clear distinction about which part is open, and which part is closed

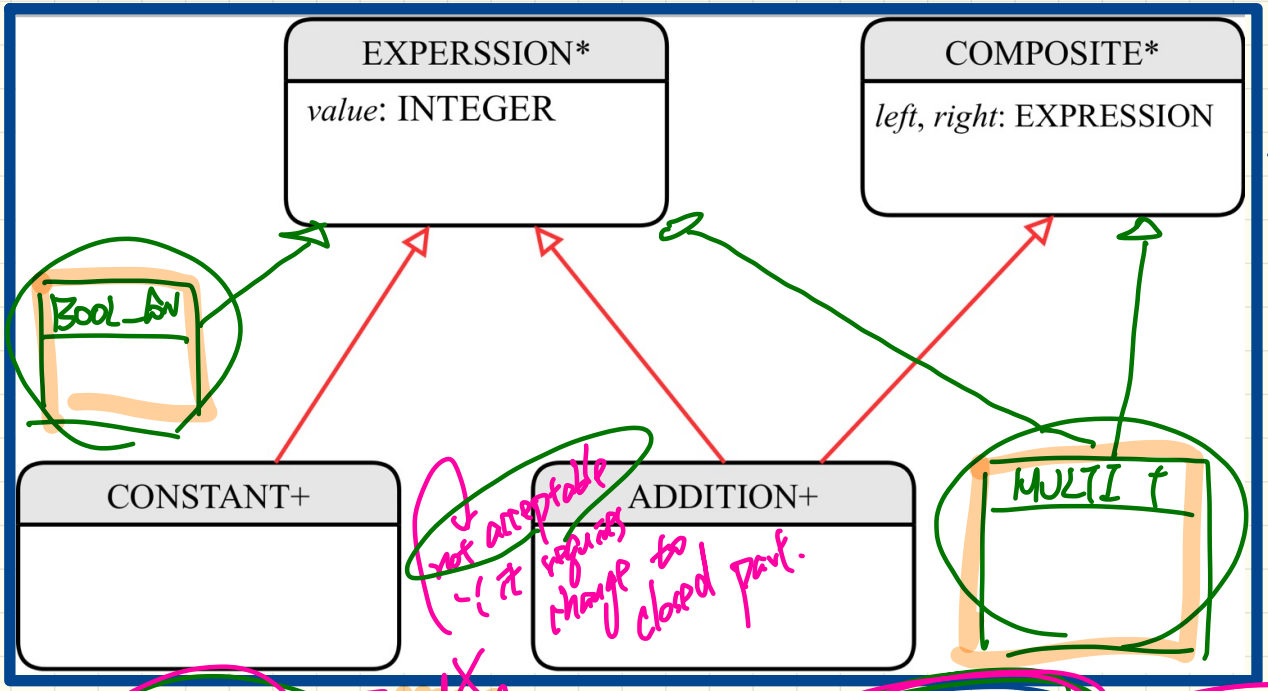
② Overtime, the open part should be touched when there's an extension.
↳ the closed part should be touched rarely (if not never)

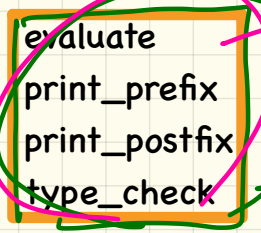
Extensions
↳ ① a new operation
e.g. (code gen).
e.g. ② a new bsn. op.
(multiplication)

Extensions ×
① supplier.
② very rarely (if not never)


Open closed principle
Visitor pattern is only applicable if one of the alternatives is satisfied.
alt 1
alt 2
language structure (+)
language operation (code_gen)
for visitor pattern
if this was judged to be the right alternative, do not use visitor.
language structure (+)
language operation (code_gen)

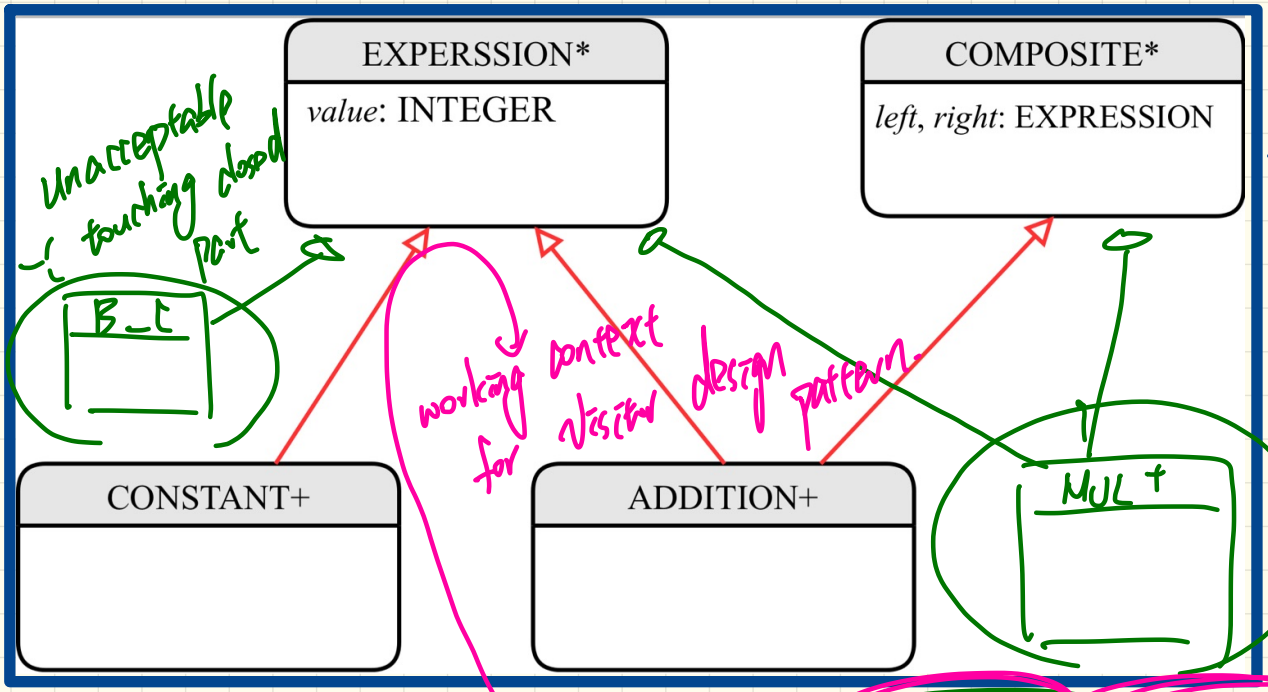
Design of a Language Application: Open-Closed Principle





	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Design of a Language Application: Open-Closed Principle



evaluate
print_prefix
print_postfix
type_check

Operations

acceptable touching open part

code_gen

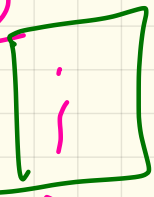
	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

apply_visitor_pattern(my_app)

require

alt 2: structure closed
operations open

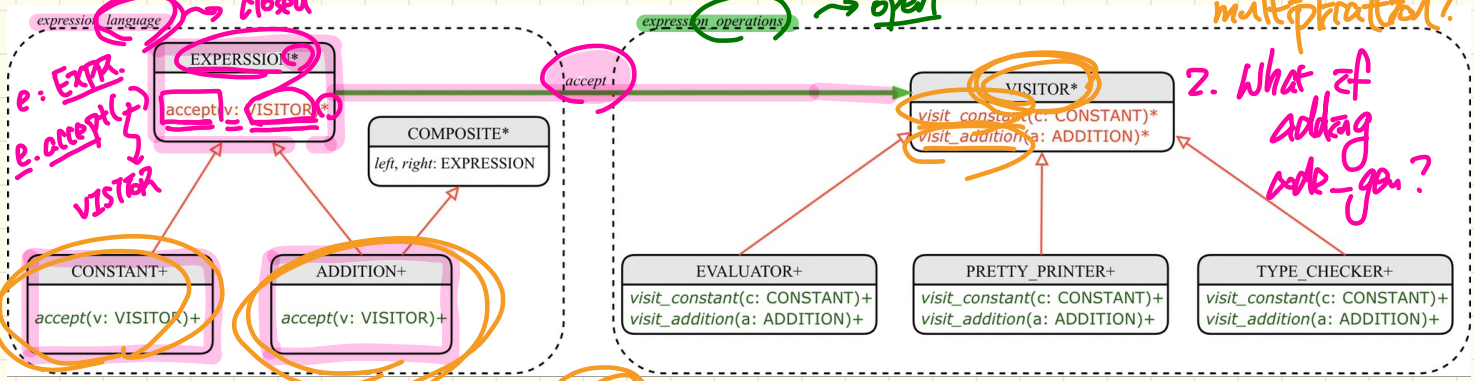
do



ensure

SCP.
cohesion.

Visitor Design Pattern: Architecture



How to Use Visitors

```
1  test_expression_evaluation: BOOLEAN
2    local add, c1, c2: EXPRESSION ; v: VISITOR
3    do
4      create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
5      create {ADDITION} add.make (c1, c2)
6      create {EVALUATOR} v.make
7      add.accept (v)
8      check attached {EVALUATOR} v as eval then
9        Result := eval.value = 3
10     end
11   end
```