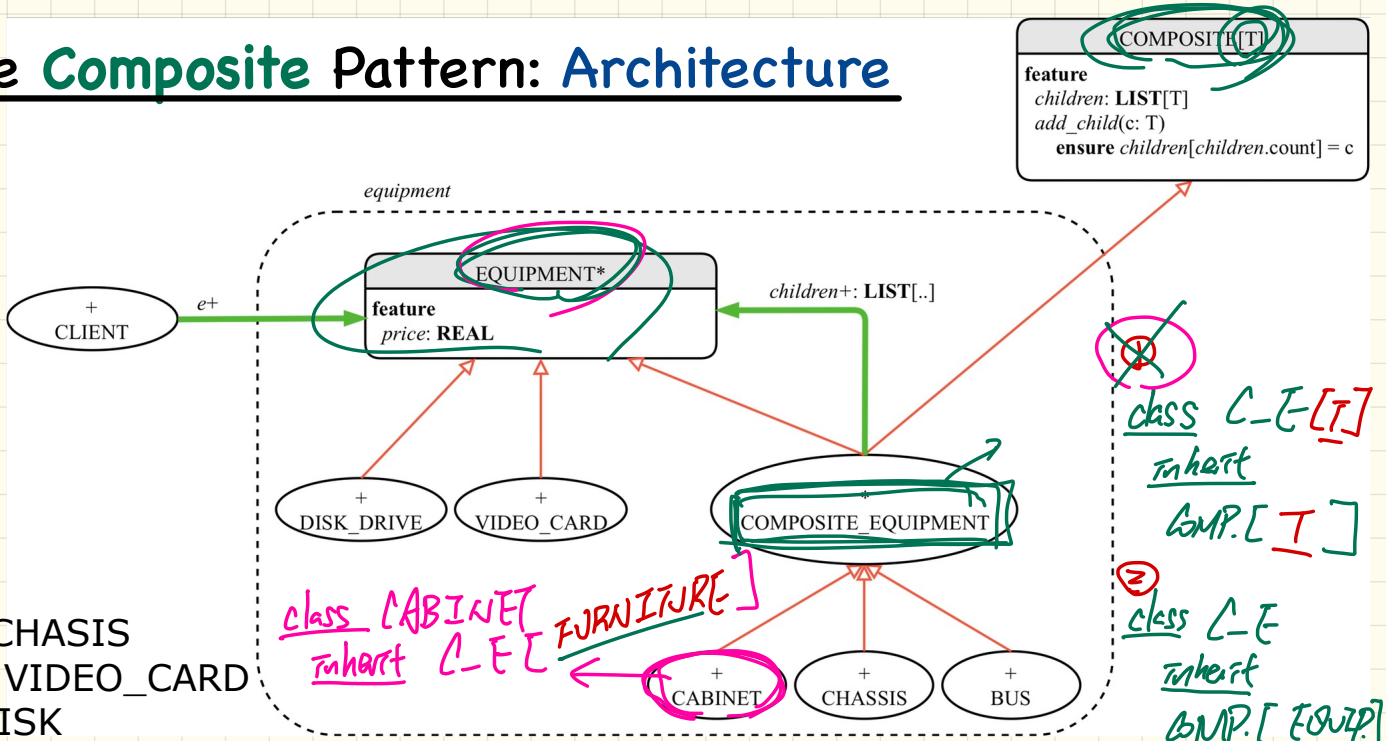
Lecture 14
Tuesday October 29

The Composite Pattern: Architecture



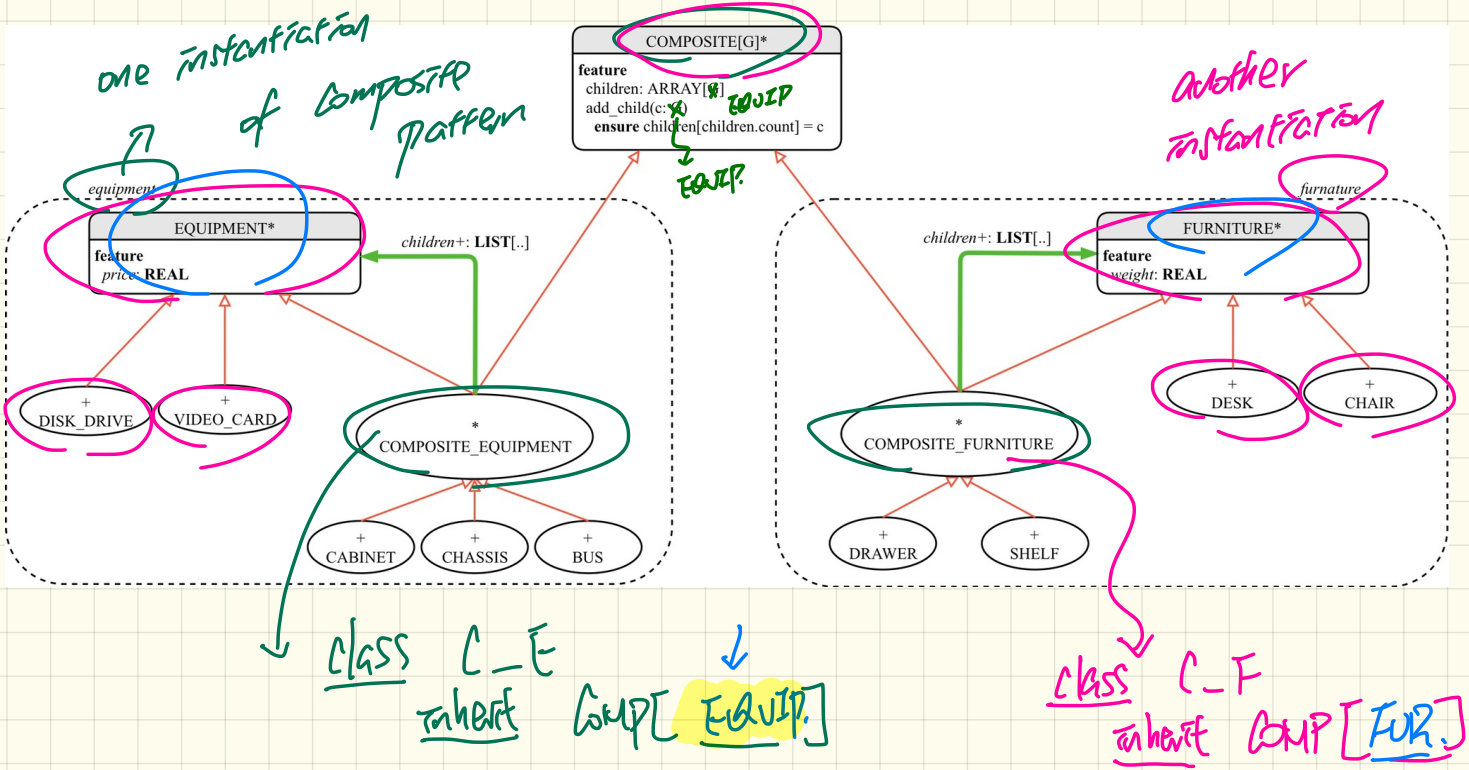
ch: CHASIS
crd: VIDEO_CARD
d: DISK
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)
crd.add_child(d)

Why is COMPOSITE a separate class?

single choice principle

The Composite Pattern: Architecture

COMPOSITE class is reusable by instances of the composite pattern.



The Composite Pattern: Implementation

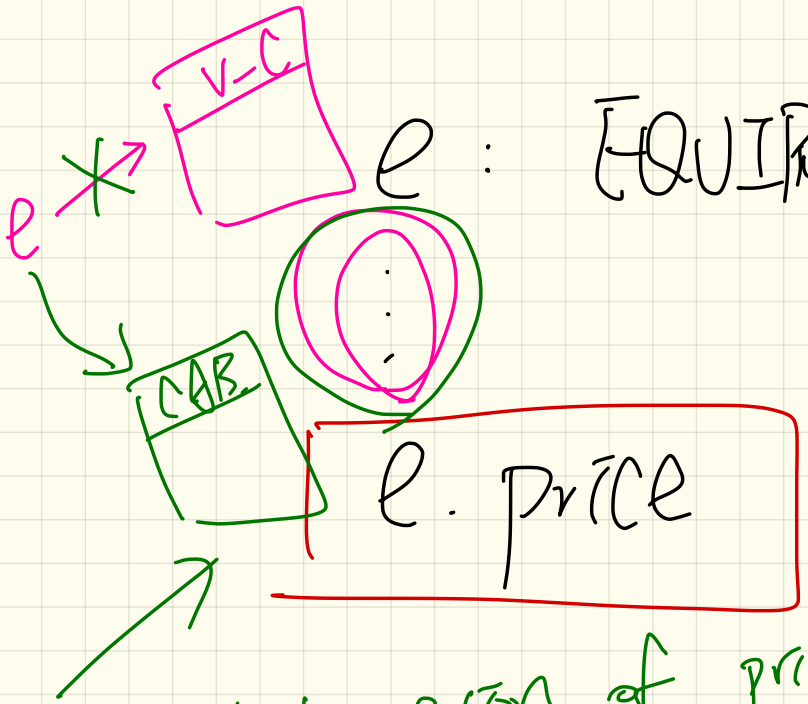
```
deferred class
  EQUIPMENT
feature
  name: STRING
  price: REAL -- uniform access principle
end
```

```
deferred class
  COMPOSITE[T]
feature
  children: LINKED_LIST[T]

  add_child (c: T)
    do
      children.extend (c) -- Polymorphism
    end
end
```

```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is an attribute
    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-equipments
    do
      across
        children as cursor
      loop
        Result := Result + cursor.item.price -- dynamic binding
      end
    end
end
```

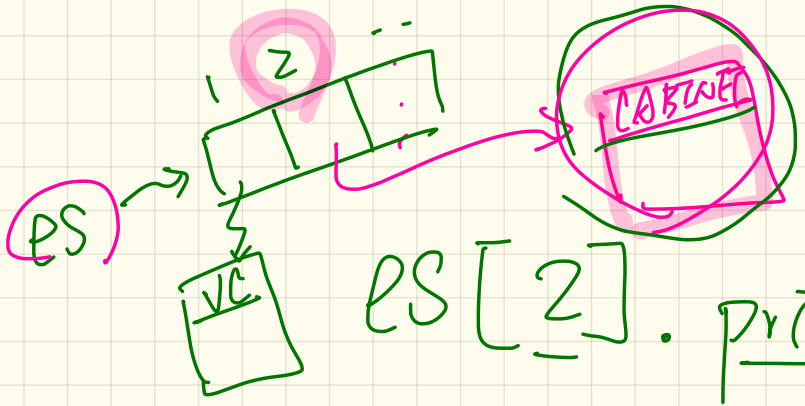


which version of price will be called?

It depends on what ... does to change e's

dynamic type.

es : ARRAY [EQUIPMENT]



es[2].price

Q1. compile? ✓

Q2 which version?
depends.

S1: EQUIP.

es[2].add_child(vc) ✗

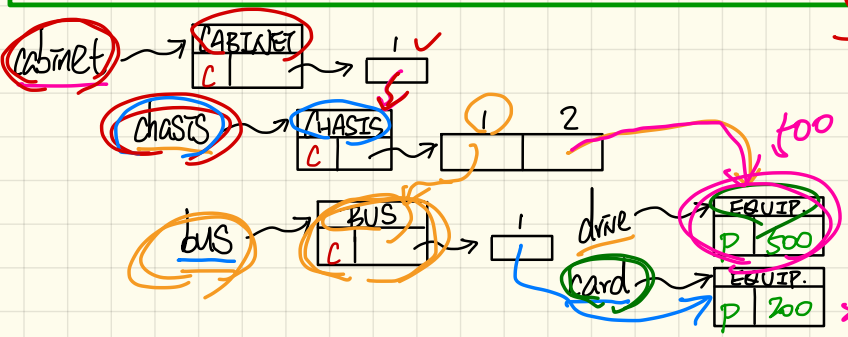
Testing the Composite Pattern

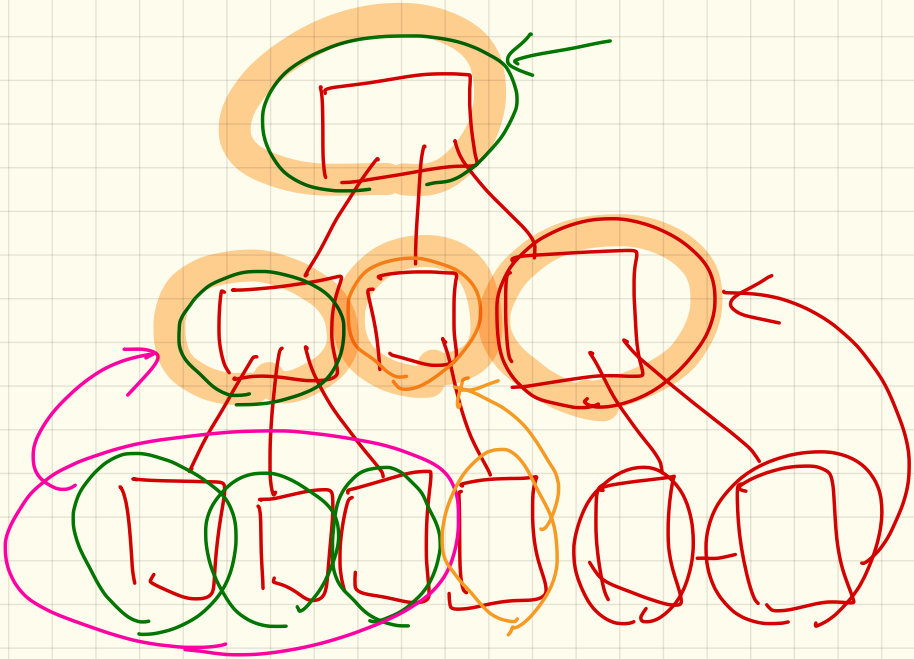
```
test_composite_equipment: BOOLEAN
  local
    card, drive: EQUIPMENT
    cabinet: CABINET -- holds a CHASSIS
    chassis: CHASSIS -- contains a BUS and a DISK_DRIVE
    bus: BUS -- holds a CARD
  do
    create {CARD} card.make("16Mbs Token Ring", 200)
    create {DISK_DRIVE} drive.make("500 GB harddrive",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

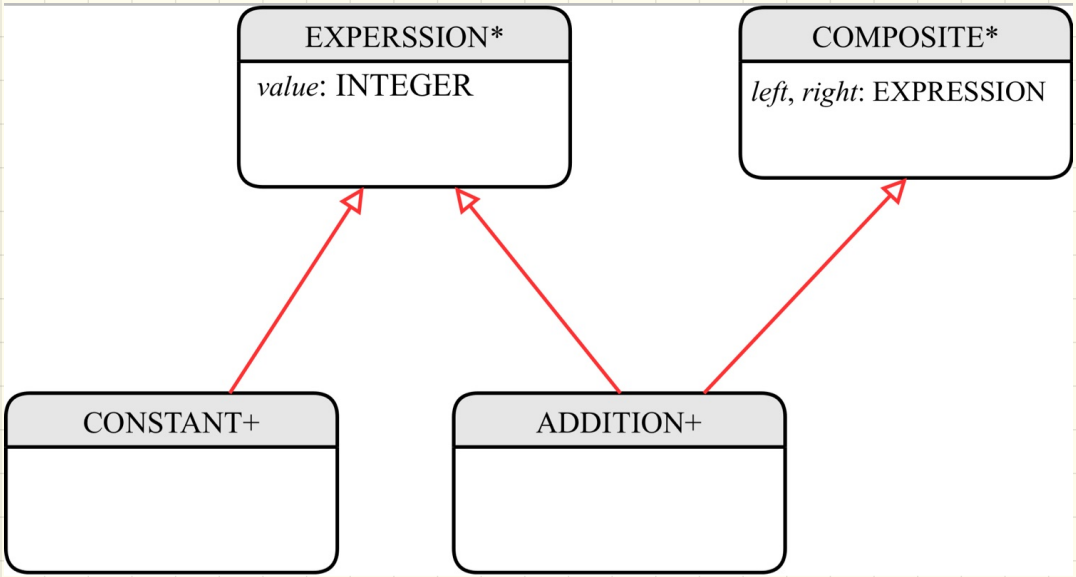
```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is
    end
end
```

```
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
```



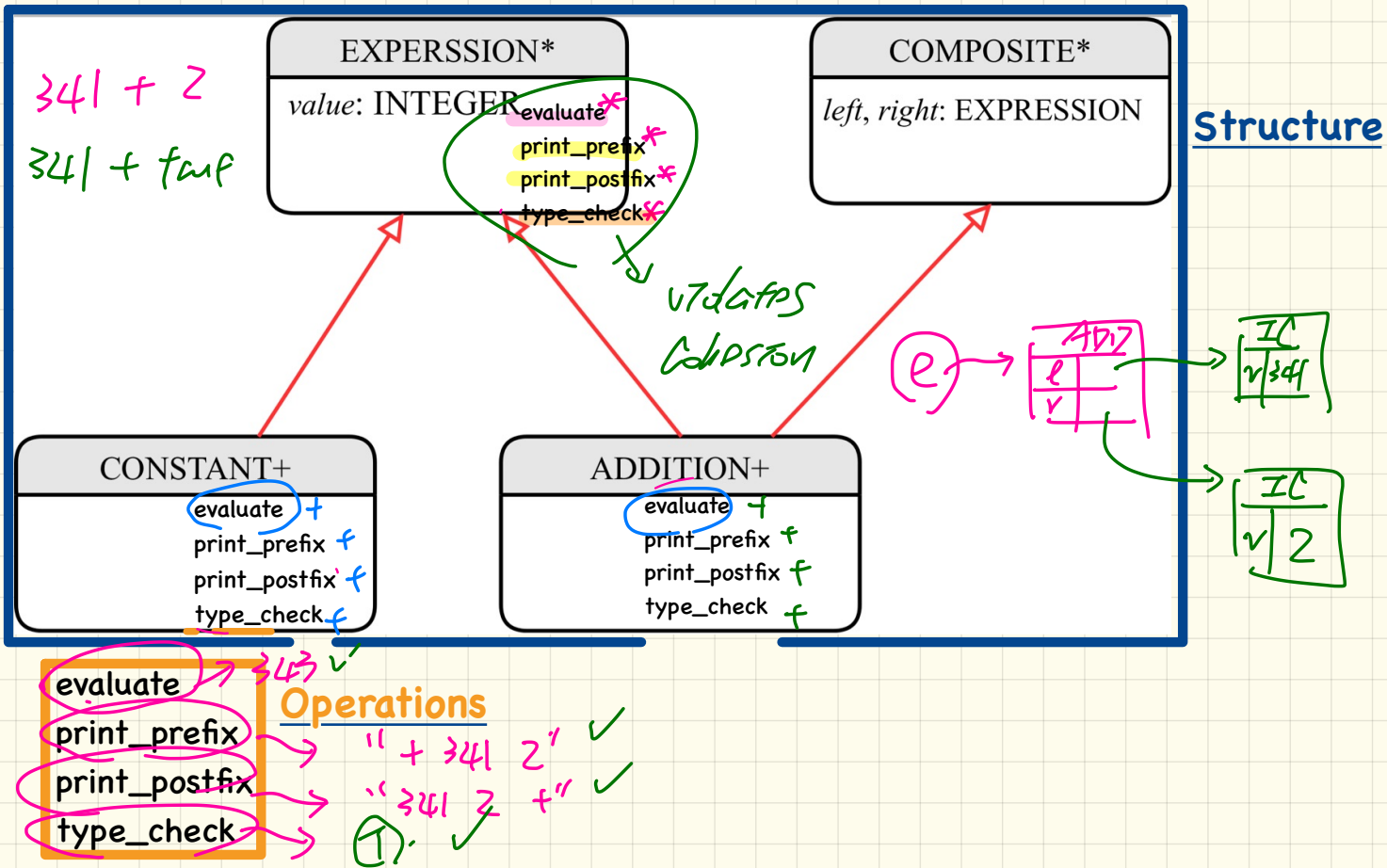


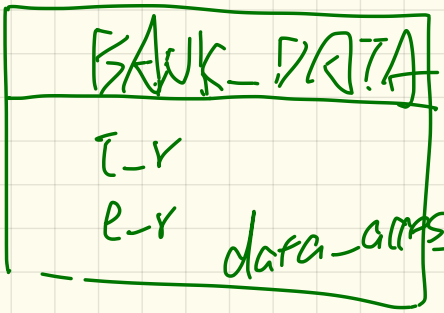
Design of Language Structure: Composite Pattern



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

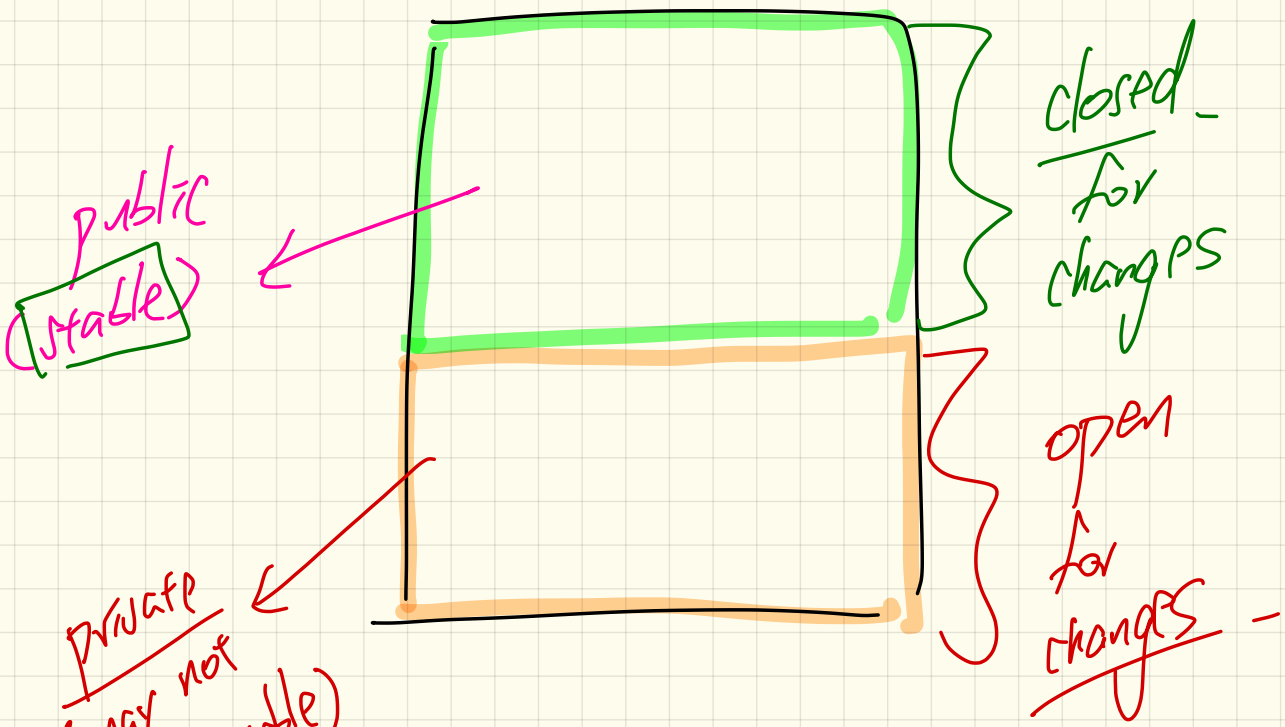
Design of Language Operation: How to Extend the Composite Pattern?



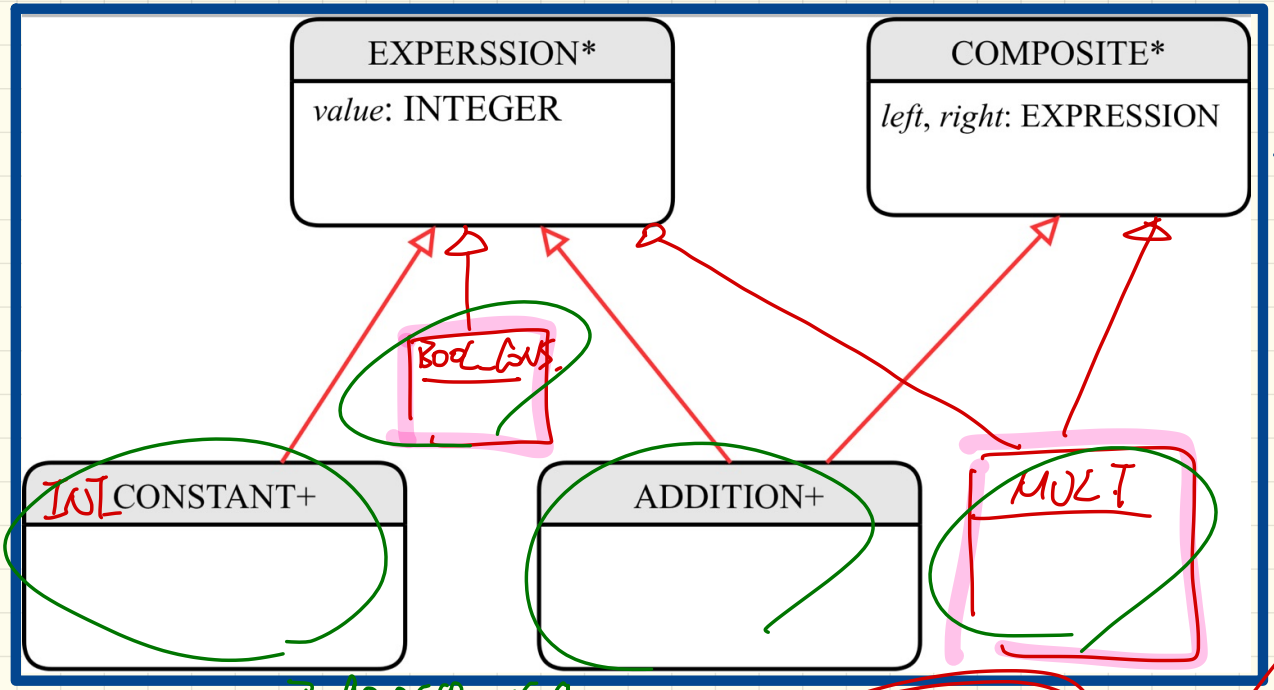


COHESION -

→ for each class only place features related to a single purpose.


public
(stable)
private
(may not
be stable)
closed
for
changes
open
for
changes

Design of a Language Application: Open-Closed Principle



Structure

evaluate
print_prefix
print_postfix
type_check

generate_java

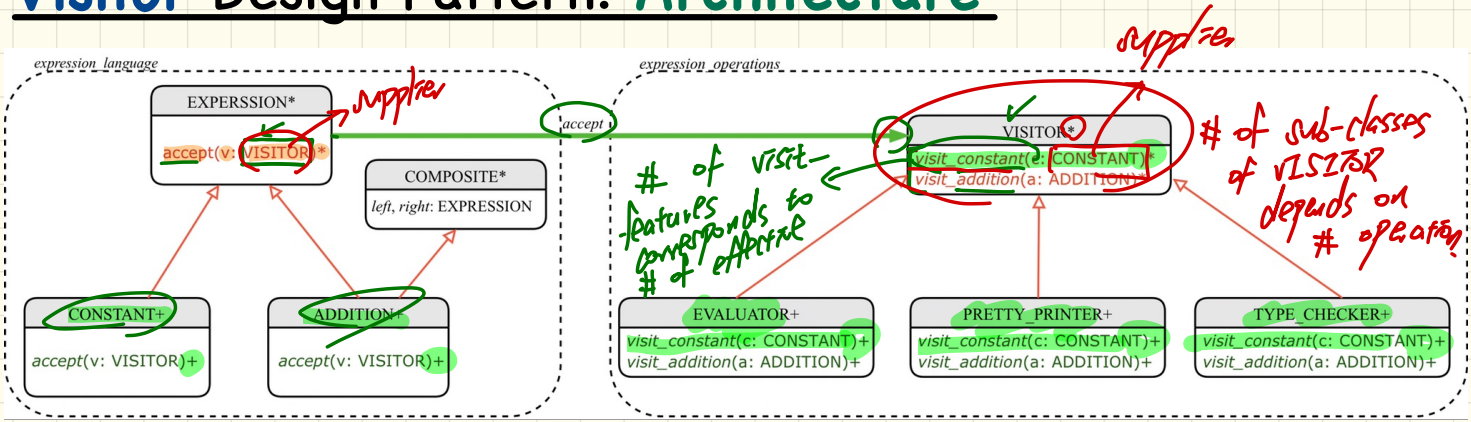
Operations

good candidate for VISITOR

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

stable

Visitor Design Pattern: Architecture



How to Use Visitors

classes in Composite

```
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
```