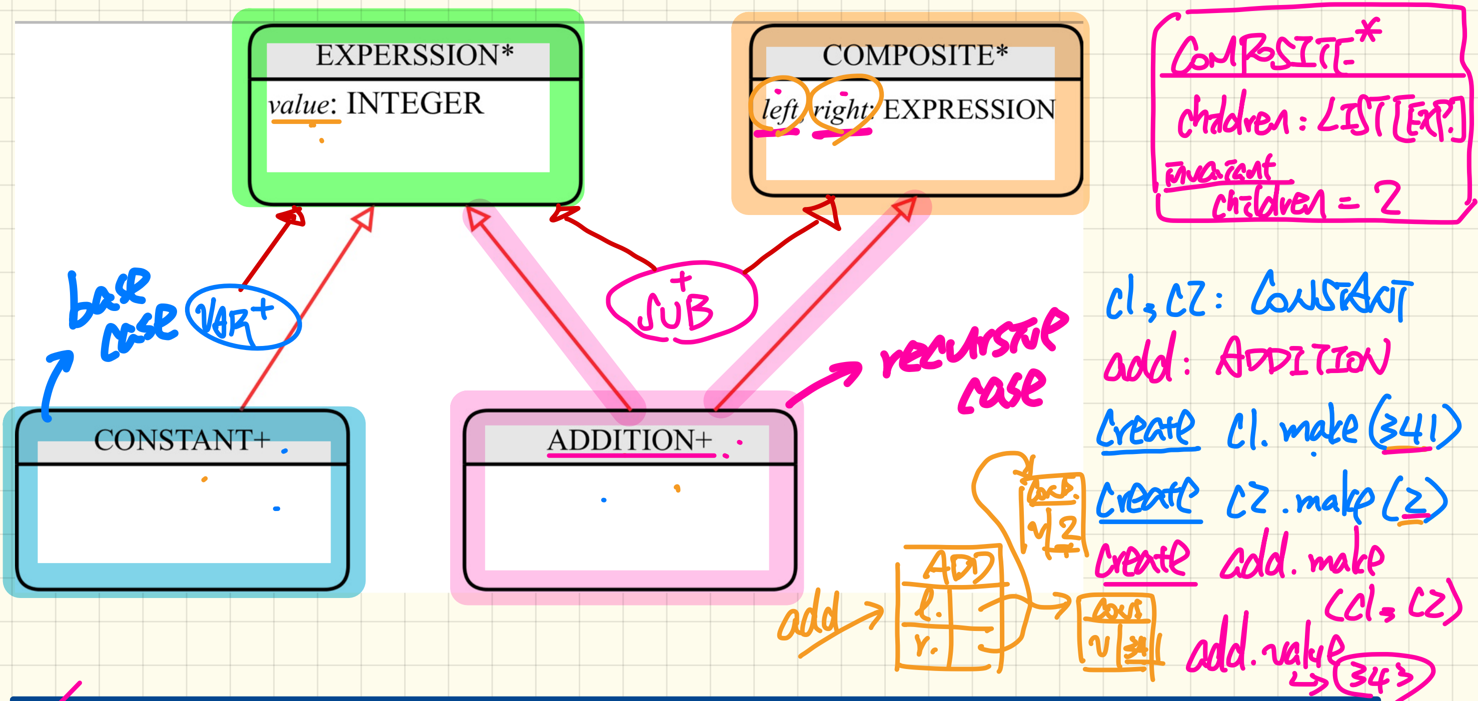
Lecture 11

Part 1

Processing Recursive Systems

Design of Language Structure: Composite Pattern



c1, c2 : CONSTANT

add : ADDITION

create c1.make(341)

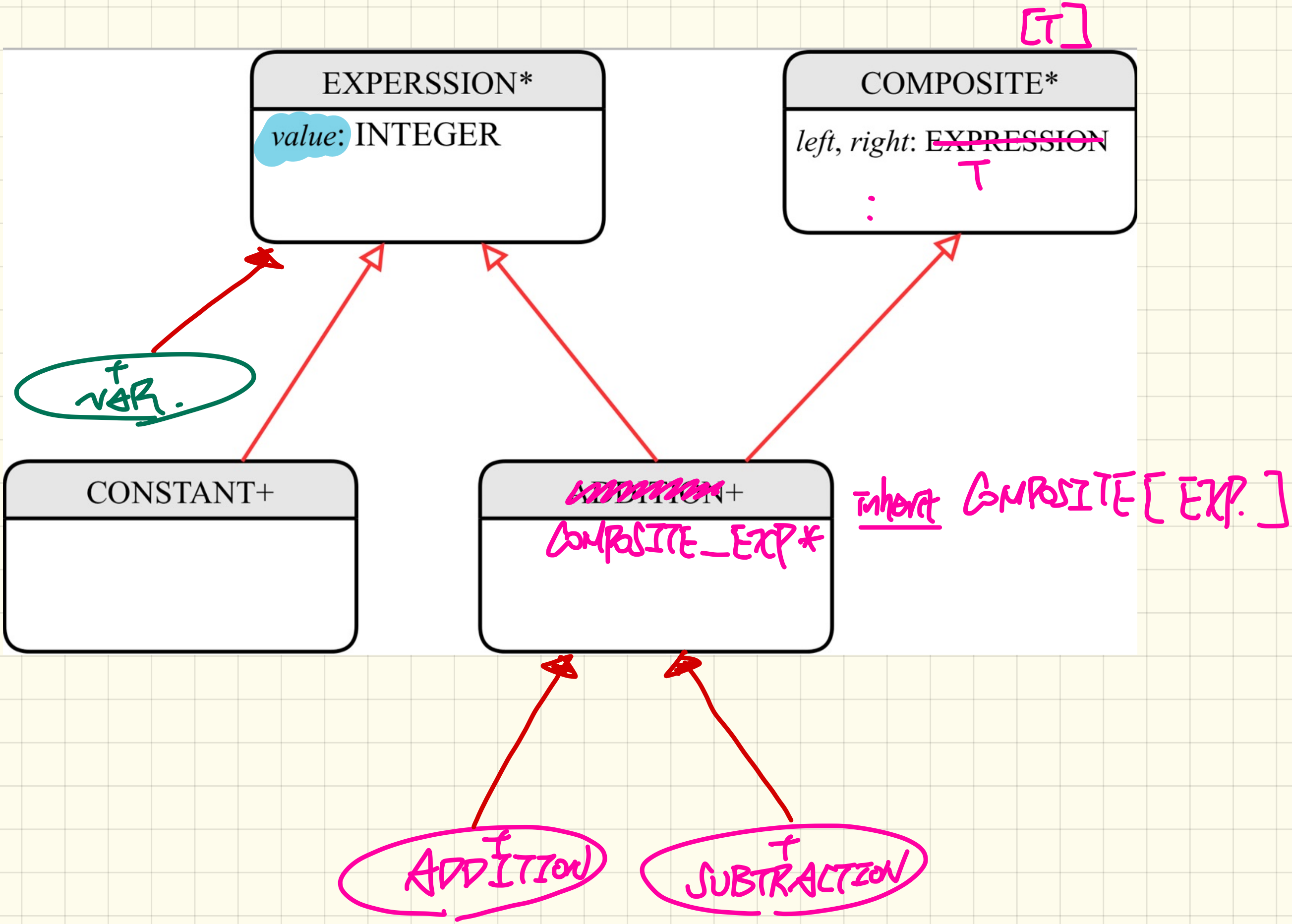
create c2.make(2)

create add.make(c1, c2)

add.value ↪ 343

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

Q: How to extend the design to include variables and subtractions?



EXPERSSION*

value: INTEGER

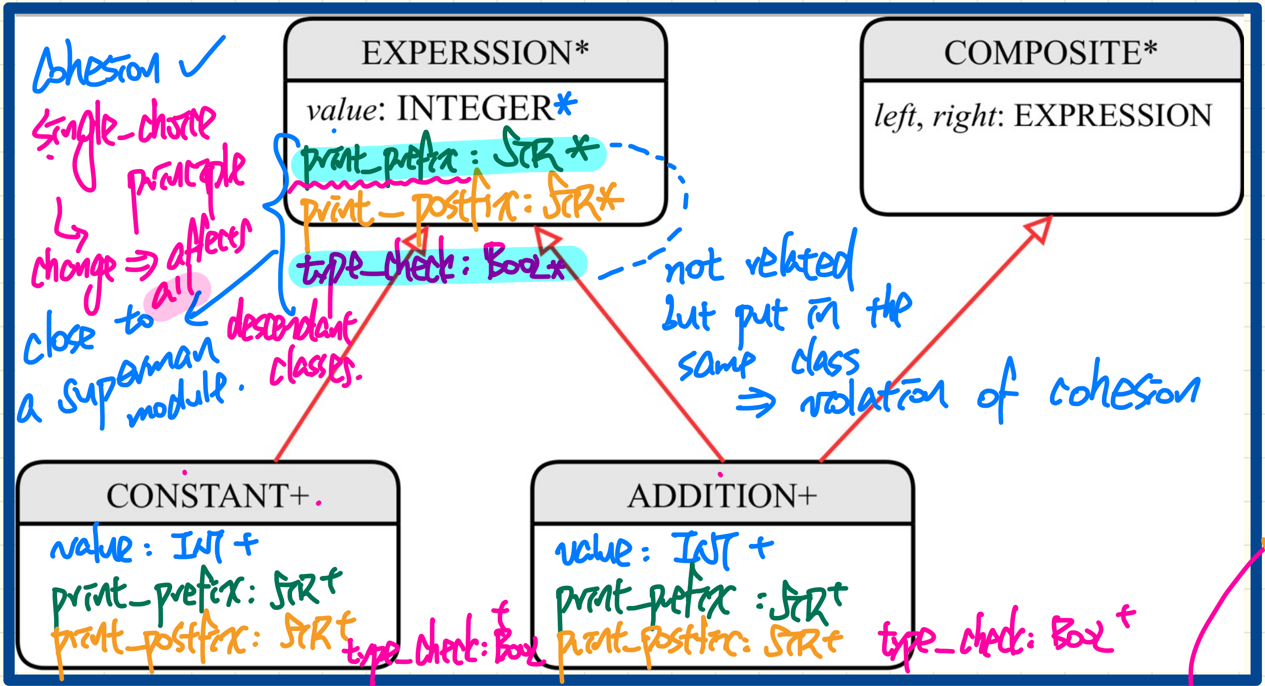
COMPOSITE*

left, *right*: ~~EXPRESSION~~

CONSTANT+

~~ADDITION~~+

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



Structure

cohesion ✓

single-choice principle

↳ change ⇒ affects all descendant classes.

close to a superman module.

not related but put in the same class

⇒ violation of cohesion

343

"+ 3 343"

"3 343 +"

true

+(3) [343]

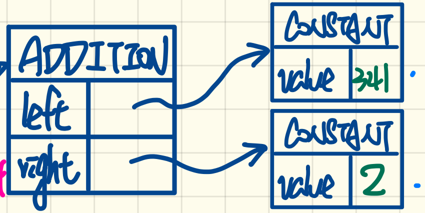
- evaluate ✓
- print_prefix ✓
- print_postfix ✓
- type_check ✓

Operations

343 + false

↳ not type correct

add



Lecture 11

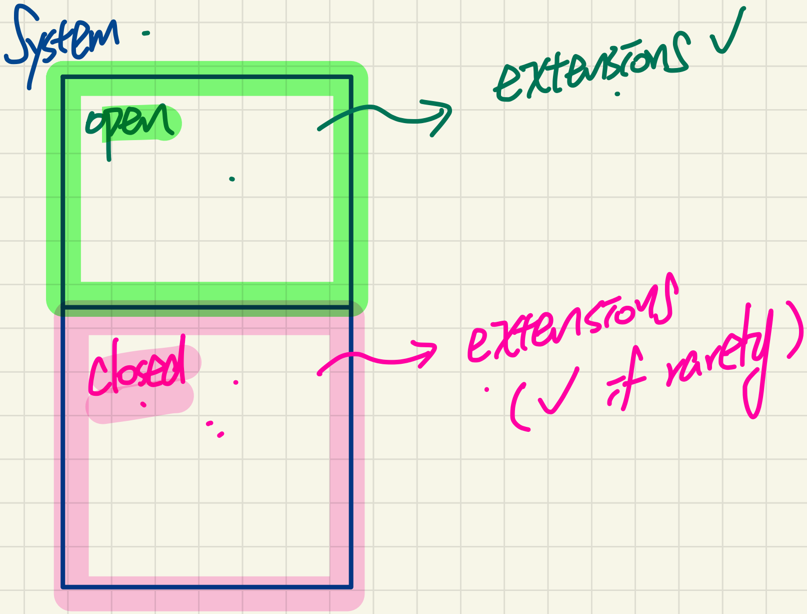
Part 2

Open-Closed Principle

Open-Closed Principle

How can the OCP be satisfied?

① There should be a clear separation/decomposition of the system into open vs. closed parts.



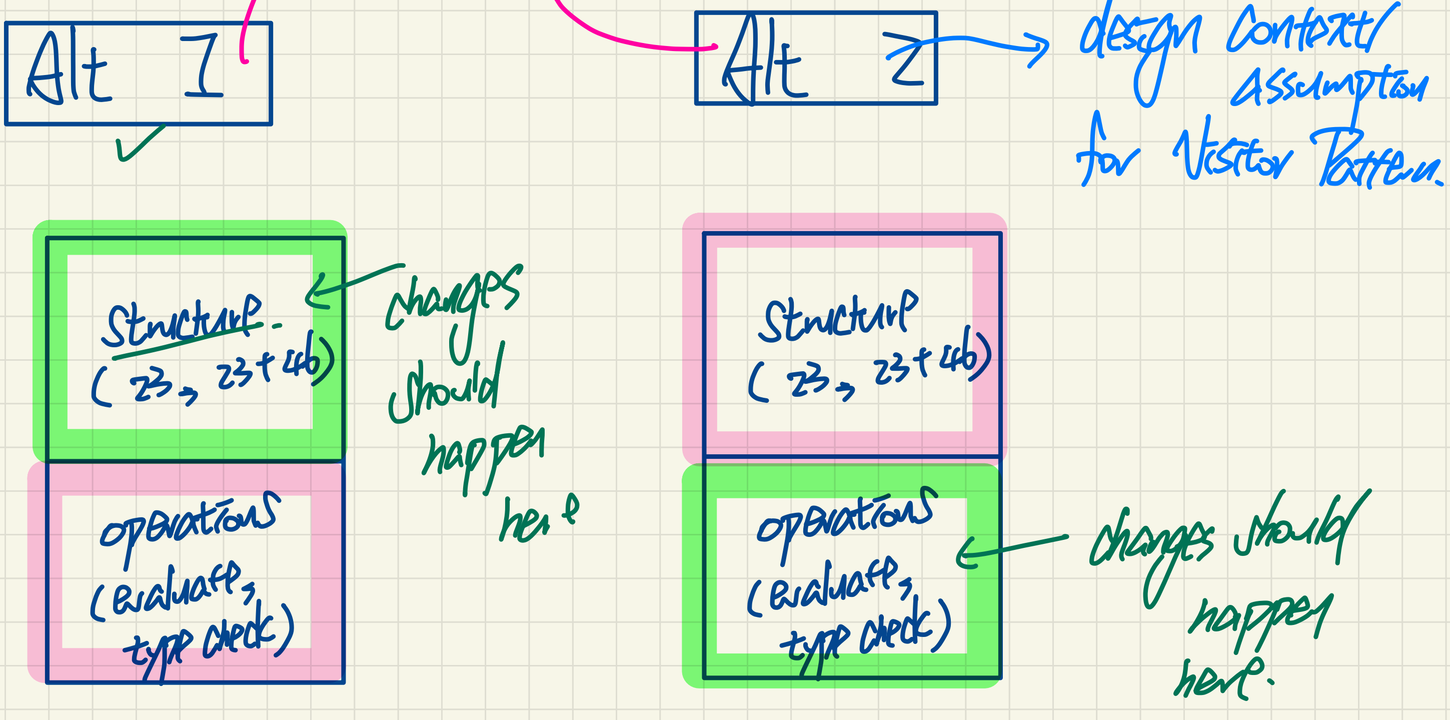
If there's a change:

② Mostly the change should touch the open part.

③ Rarely, if at all, a change may have to touch the closed part.

both alternatives satisfy OCP.

Applying the OCP to Exp. language design.



Alt I ✓

Alt 2 → design context/assumption for Visitor Pattern.

Structure (23, 23+46)

operations (evaluate, type check)

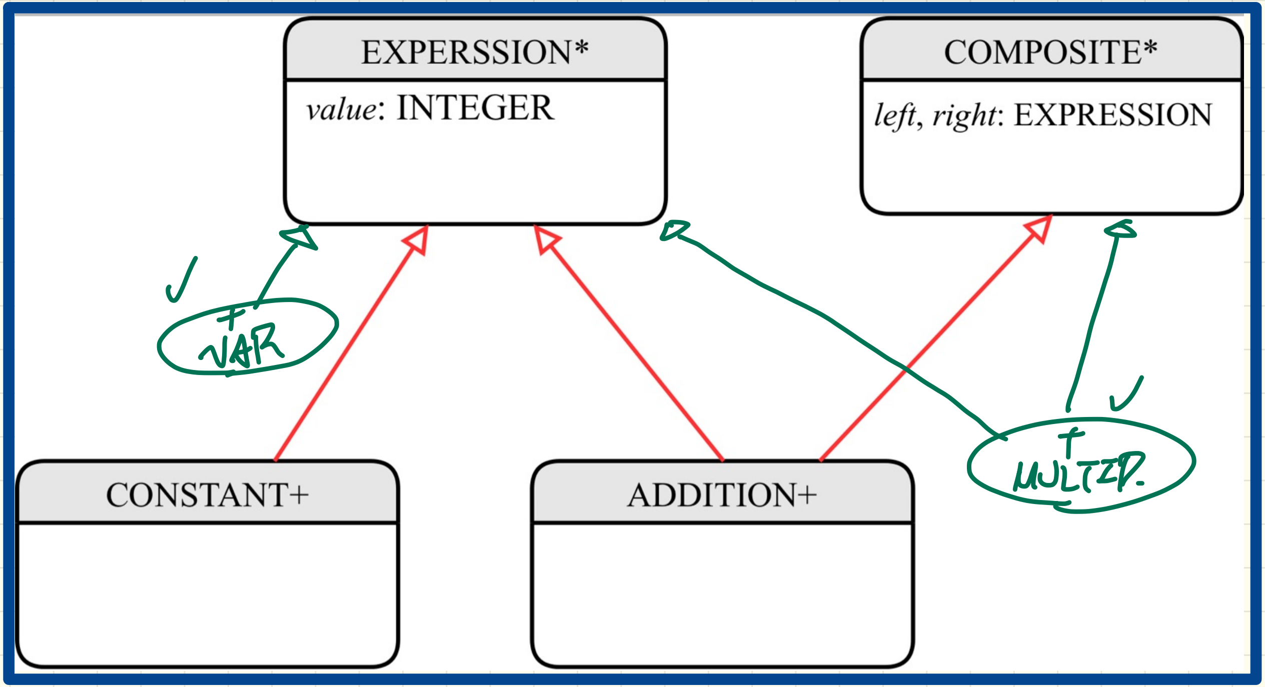
changes should happen here

Structure (23, 23+46)

operations (evaluate, type check)

changes should happen here.

Design of a Language Application: Open-Closed Principle



EXPERSSION*

value: INTEGER

COMPOSITE*

left, *right*: EXPRESSION

CONSTANT+

ADDITION+

Structure

→ evaluate ·

print_prefix.

print_postfix.

type_check·

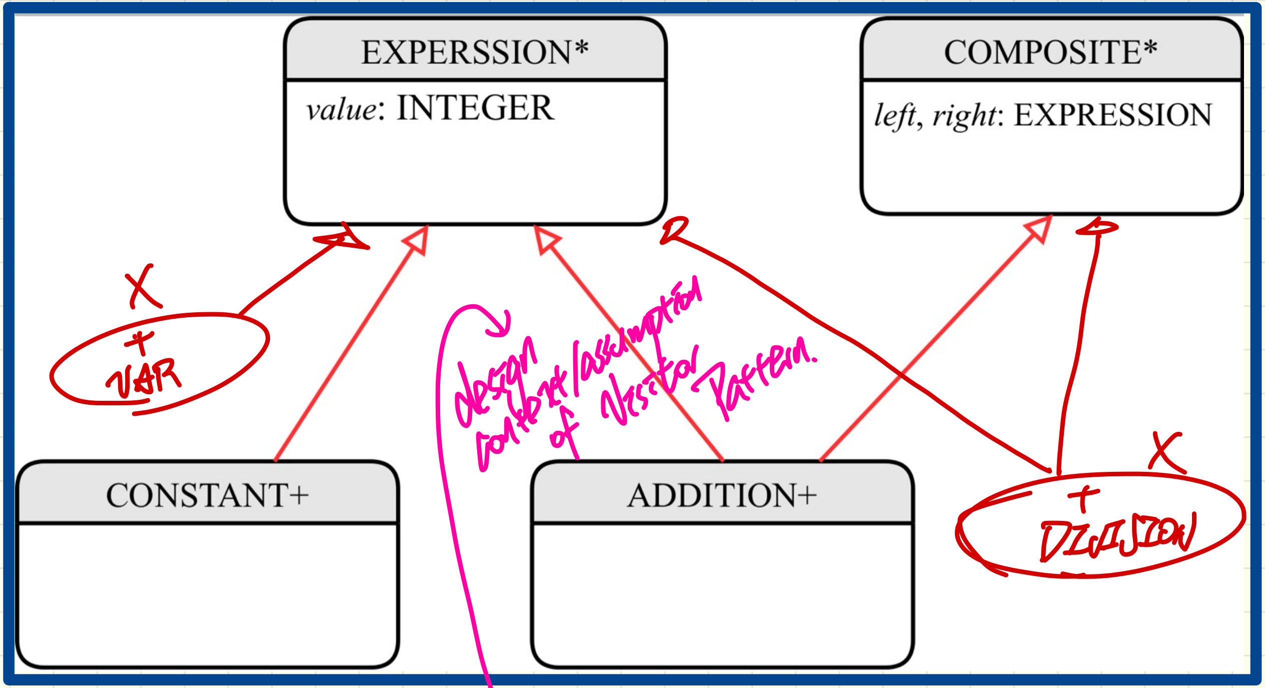
Operations

× optimize ←

× code_gen. ←

	Structure	Operations
Alternative 1	Open	→ Closed ←
Alternative 2	Closed	Open

Design of a Language Application: Open-Closed Principle



Structure

- evaluate
- print_prefix
- print_postfix
- type_check

Operations

code-gen ✓
optimize ✓

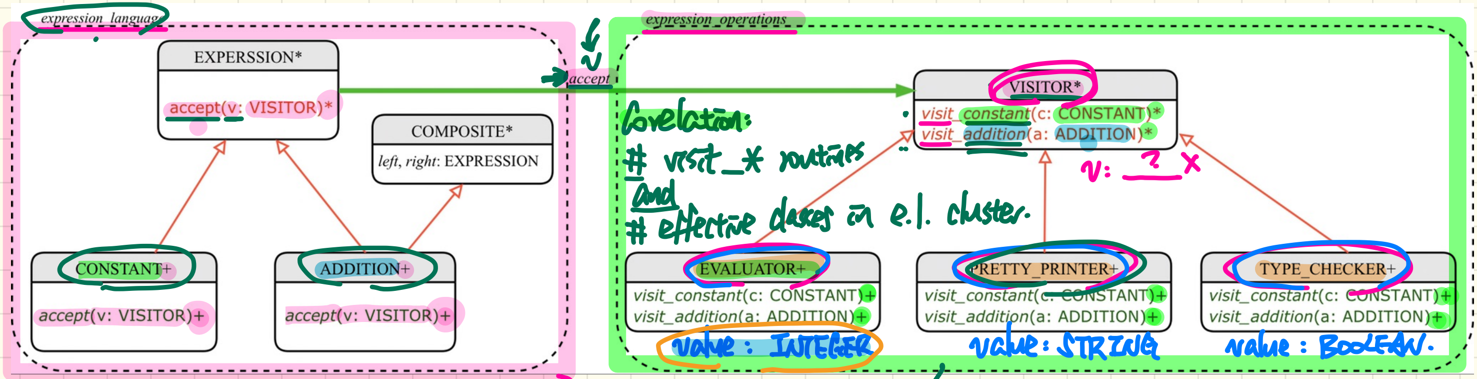
	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Lecture 11

Part 3

Visitor Design Pattern

Visitor Design Pattern: Architecture



closed

open

without a cast

v. value ×

∵ value not declared in VISITOR.

1+2+3

1+2.

How to Use Visitors

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

ST.

DT? E. P.P. T.C.

visitor 'v' will visit 'add' automatically.

add . accept (v)

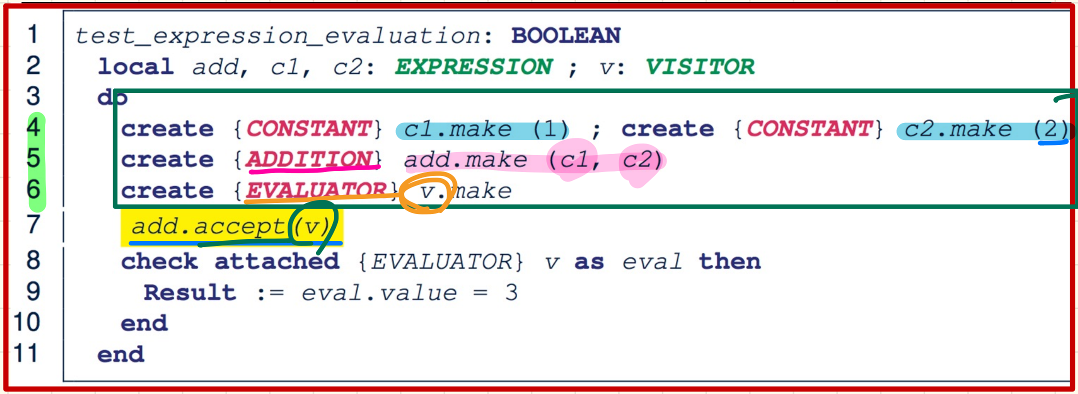
composite object.

visitor object

```
create {P_P} v.make
add.accept(v)
```

```
check {P_P} v as PP then
  R := PP.value ~ "1+2"
end
```

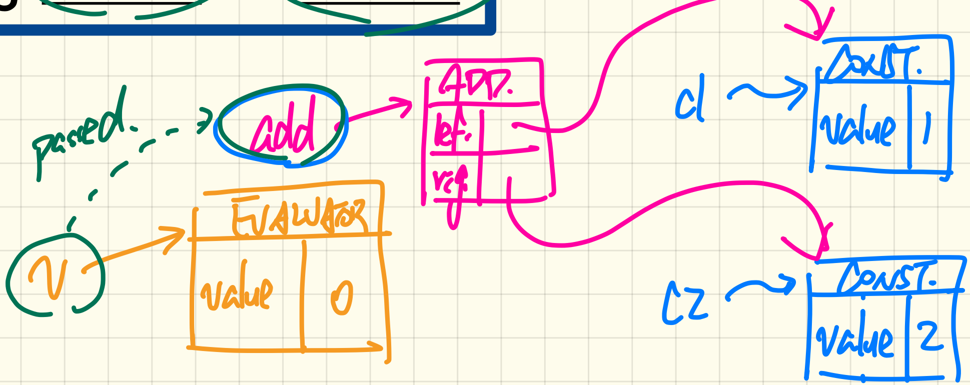
Visitor Design Pattern: Implementation



```
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
```

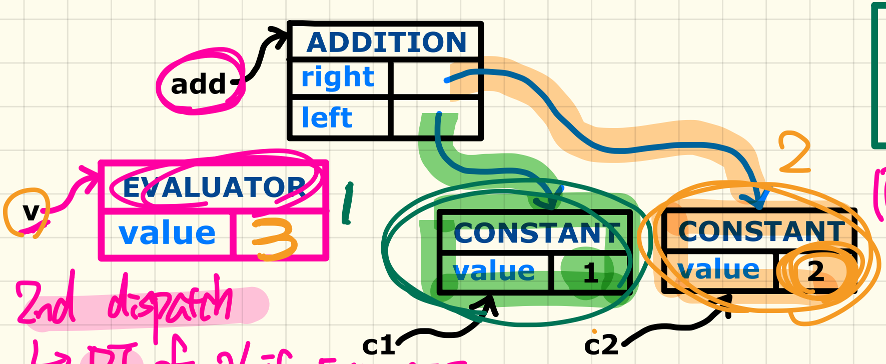
→ building the composite/recursive object.

Visualizing Line 4 to Line 6



Executing **Composite** and **Visitor** Patterns at **Runtime**

Tracing add.accept(v)
Double Dispatch



2nd dispatch
↳ DT of v is EVALUATOR
↳ visit_addition in EVALUATOR called

add.accept(v)
↳ 1st (dynamic) dispatch
↳ DT of add is ADDITION
↳ accept in ADDITION called

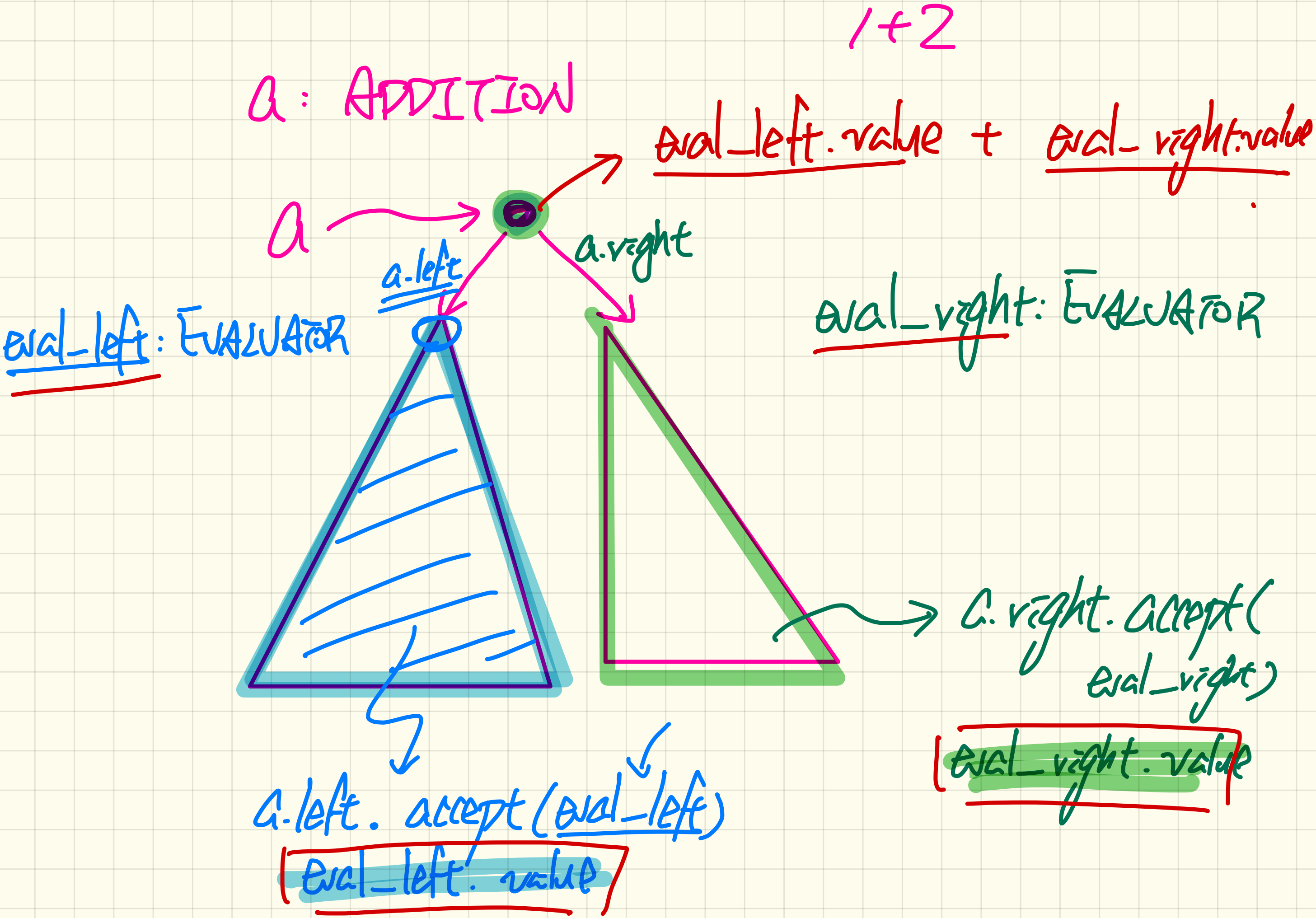
```
deferred class VISITOR
  visit_constant(c: CONSTANT) deferred end
  visit_addition(a: ADDITION) deferred end
end
```

```
class EVALUATOR inherit VISITOR
  value : INTEGER
  visit_constant(c: CONSTANT) do value := c.value end
  visit_addition(~~a: ADDITION~~) add
   local eval_left, eval_right: EVALUATOR
  do a.left.accept(eval_left)      → double dispatch
     a.right.accept(eval_right)    → double dispatch
     value := eval_left.value + eval_right.value
                  1           +        2          = 3
  end
end
```

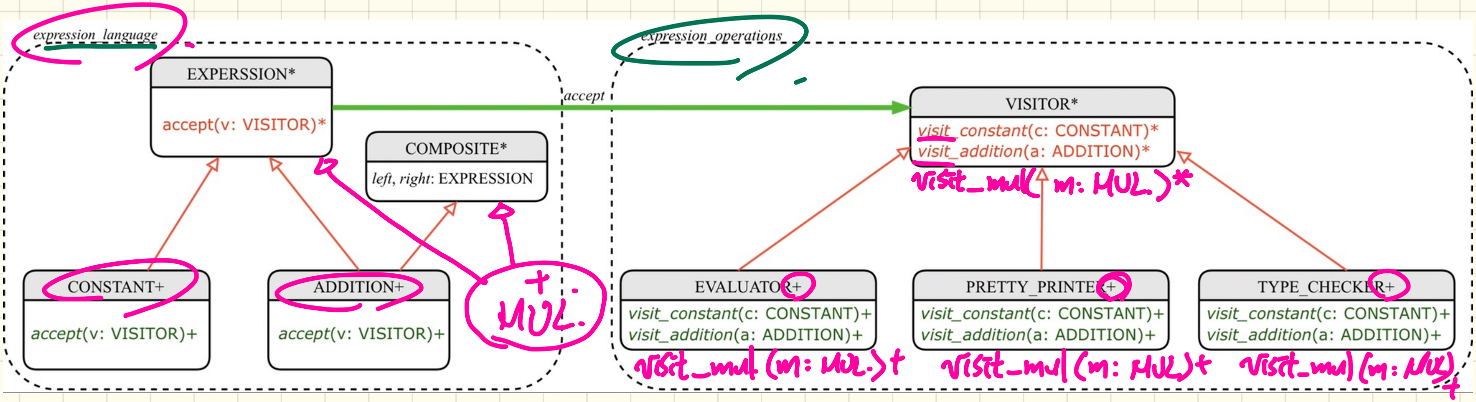
exercise: explain

```
class CONSTANT inherit EXPRESSION
...
  accept(v: VISITOR)
   do
     v.visit_constant(Current)
                      CONSTANT
   end
end
```

```
class ADDITION
inherit EXPRESSION COMPOSITE
         v     ST.
...
  accept(~~v: VISITOR~~)
   do                          add
     v.visit_addition(Current)
                      ADDITION
   end
end
```


1+2
a : ADDITION
eval_left.value + eval_right.value
a
a.left
a.right
eval_left : EVALUATOR
eval_right: EVALUATOR
a.right.accept(eval_right)
eval_right.value
a.left. accept(eval_left)
eval_left. value

Visitor Pattern: Open-Closed and Single-Choice Principles



expression_language

EXPERSSION*

accept(v: VISITOR)*

COMPOSITE*

left, *right*: EXPRESSION

CONSTANT+

accept(v: VISITOR)+

ADDITION+

accept(v: VISITOR)+

+ MUL.

accept

expression_operations

VISITOR*

visit_constant(c: CONSTANT)*

visit_addition(a: ADDITION)*

visit_mul(m: MUL)*

EVALUATOR+

visit_constant(c: CONSTANT)+

visit_addition(a: ADDITION)+

visit_mul (m: MUL)+

PRETTY_PRINTER+

visit_constant(c: CONSTANT)+

visit_addition(a: ADDITION)+

visit_mul (m: MUL)+

TYPE_CHECKER+

visit_constant(c: CONSTANT)+

visit_addition(a: ADDITION)+

visit_mul (m: MUL)+

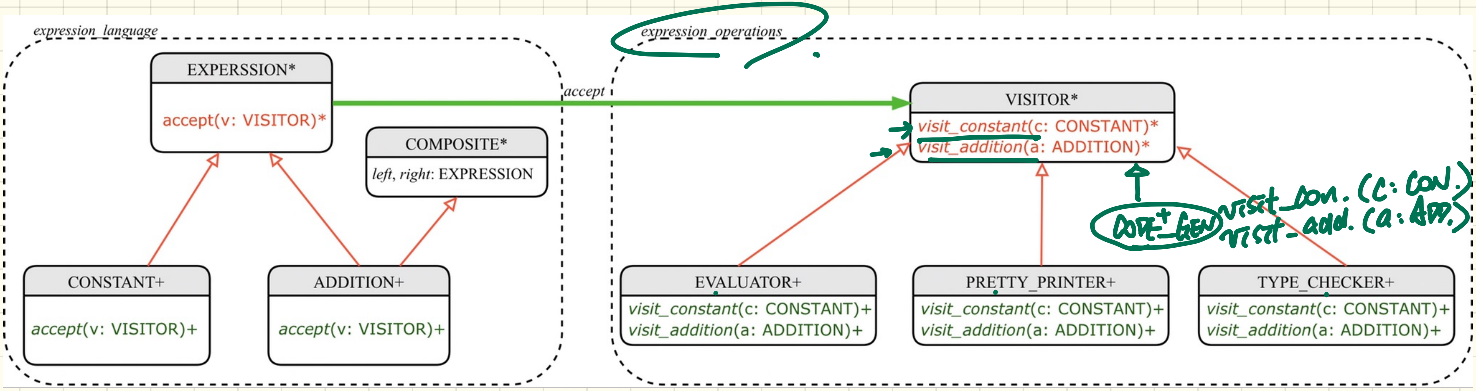
What if a new language construct is added?

① single class to be added to the structure
② multiple places to modify in operations
→ violates SCP.

If the visitor pattern is adopted, what should be closed?

Structure.

Visitor Pattern: Open-Closed and Single-Choice Principles



expression_language

EXPERSSION*

accept(v: VISITOR)*

COMPOSITE*

left, *right*: EXPRESSION

CONSTANT+

accept(v: VISITOR)+

ADDITION+

accept(v: VISITOR)+

accept

expression_operations

VISITOR*

visit_constant(c: CONSTANT)*

visit_addition(a: ADDITION)*

EVALUATOR+

visit_constant(c: CONSTANT)+

visit_addition(a: ADDITION)+

PRETTY_PRINTER+

visit_constant(c: CONSTANT)+

visit_addition(a: ADDITION)+

TYPE_CHECKER+

visit_constant(c: CONSTANT)+

visit_addition(a: ADDITION)+

satisfies SCP.

What if a new language operation is added?

① single class added to operations

② all changes are restricted to this single class

If the visitor pattern is adopted, what should be open?

operations.