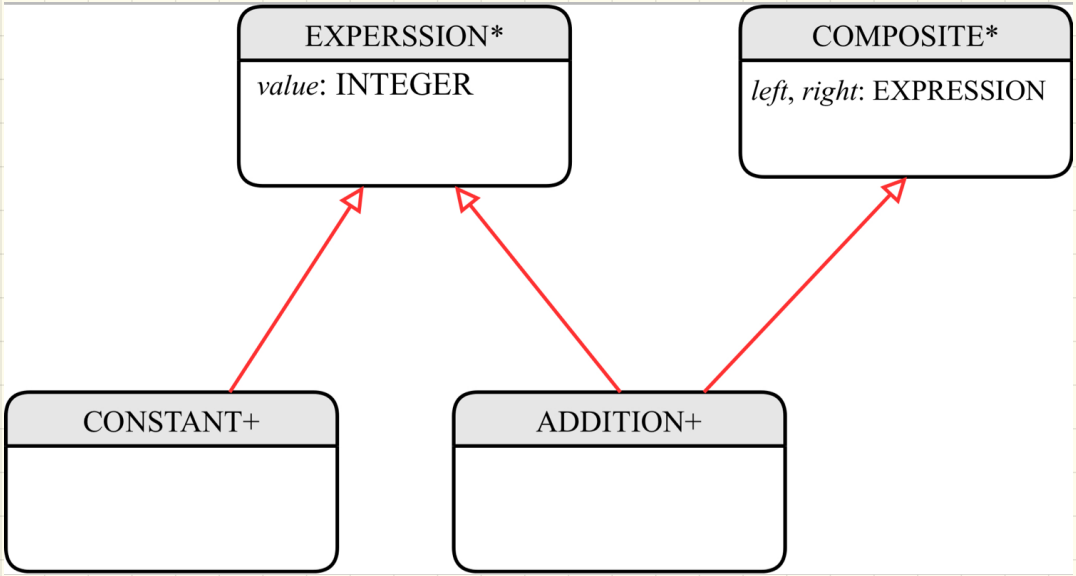
Monday March 11

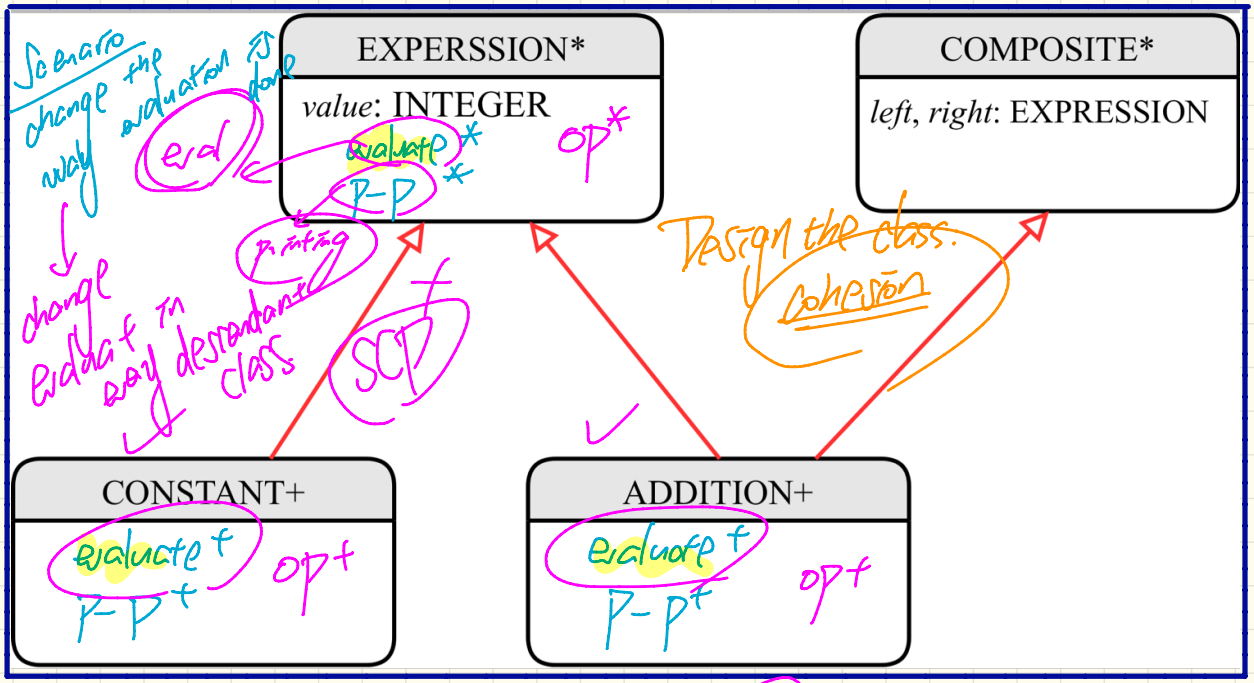
Lecture 16

Design of Language Structure : Composite Pattern



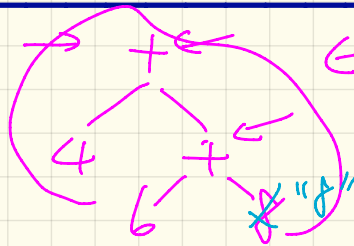
Q: How do you construct an object representing "34 + 2" ?

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



Operations: evaluate, print_prefix, print_postfix, type_check

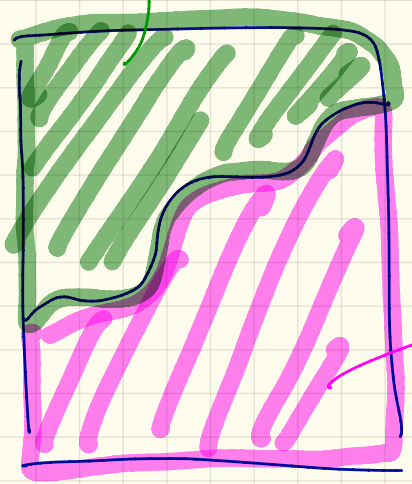
Operations



+ 4 + 6 8

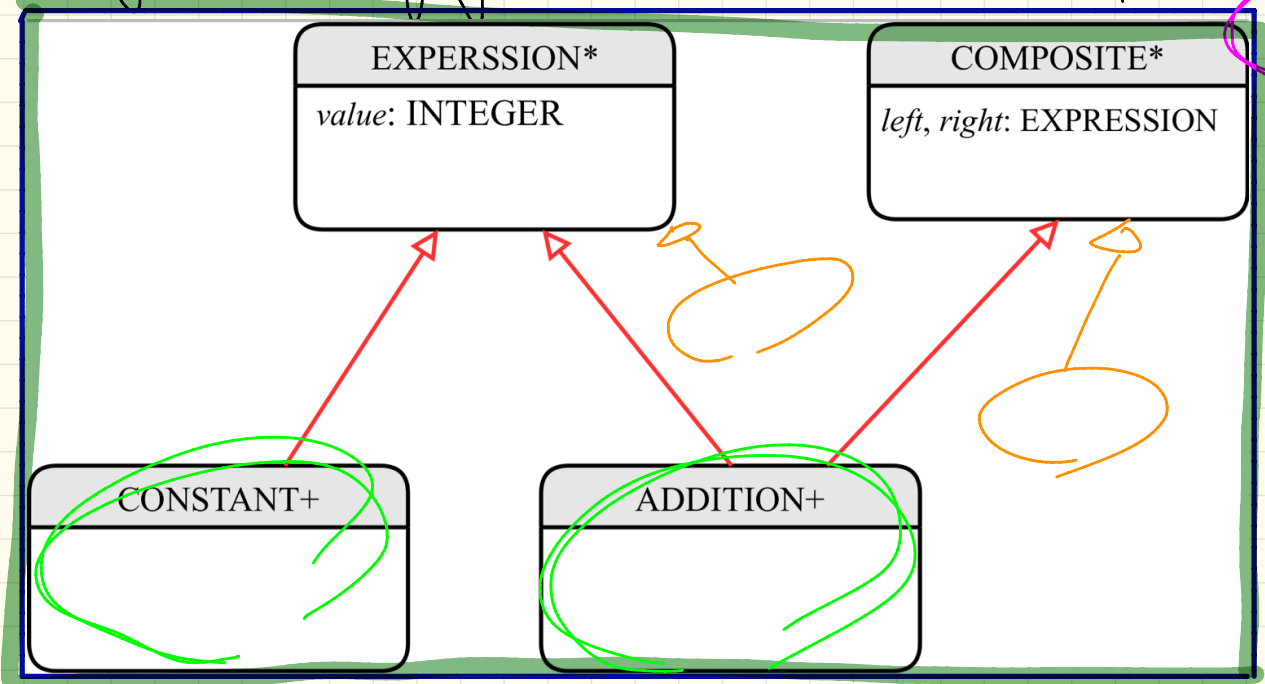
4 + 6 8 +

open for extension



closed for modification

Design of a Language Application : Open-Closed Principle



Structure

both open
↳ confusing what to extend

both closed
↳ not flexible

Operations: evaluate
print_prefix
print_postfix
type_check

Operations
generate_assembly

	Structure	Operations
Alt. 1	open	closed
Alt. 2	closed	open → visitor

Visitor

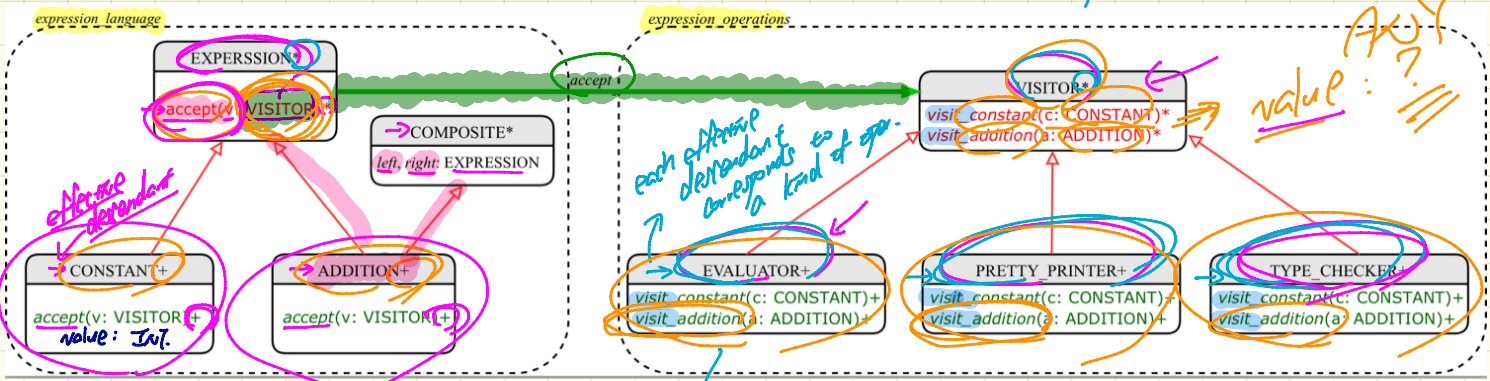
open-closed principle

open part : operations

closed part : structure

Visitor Design Pattern: Architecture

visit_expression X



list of visit_ features correspond to the type of effective descendants of EXPRESSION.

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
```

v is descendant of VIS

build the Composite tree

N. value

Client of Visitor

1. e : EXPRESSION → deferred

build the composite tree

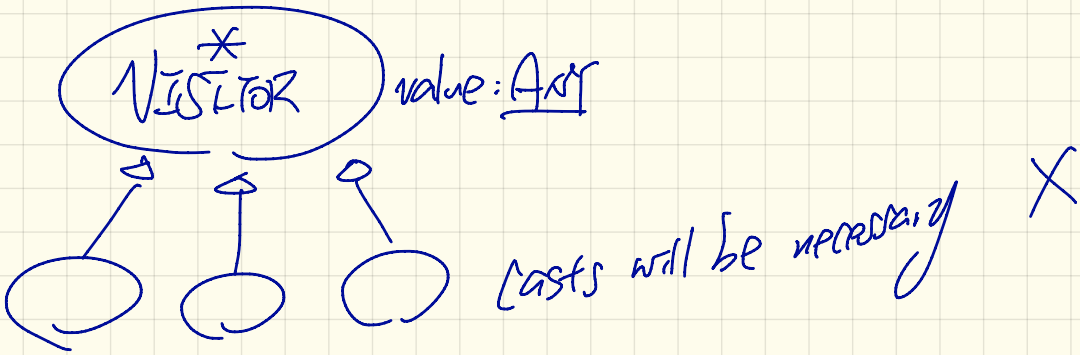
2. v : VISITOR → deferred

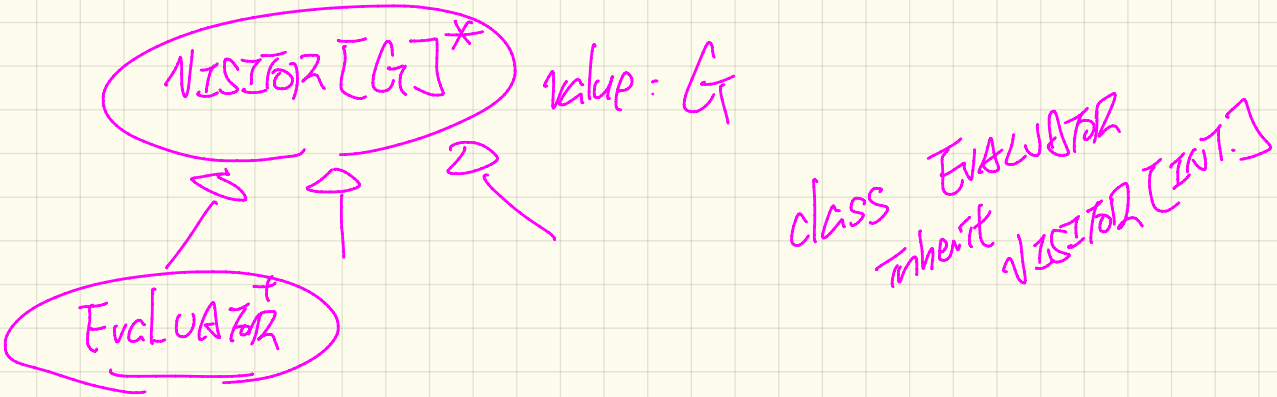
attach v to a particular VISITOR type

3. e.accept(v)

4. retrieve the result of visit from v.

Poor Design.

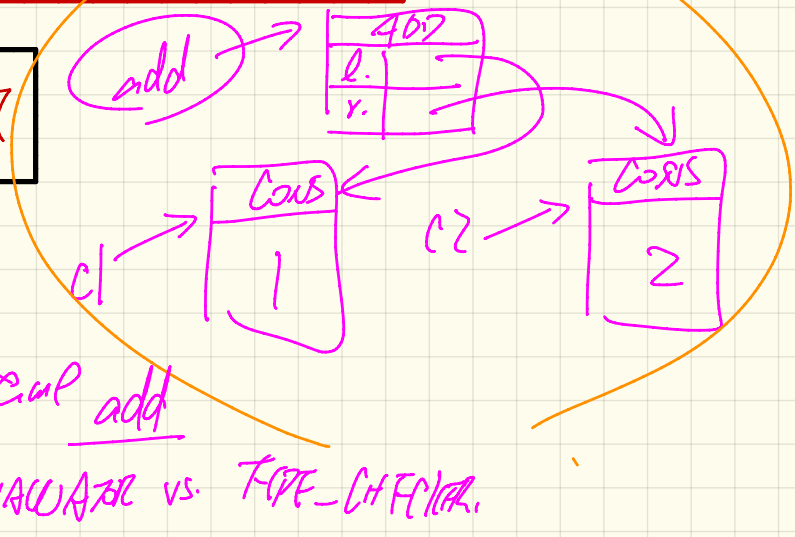

VISITOR
value: Any
casts will be necessary
X


VISITOR[G]
value: G
EVALUATOR
class EVALUATOR
inherit VISITOR[INT]

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
```

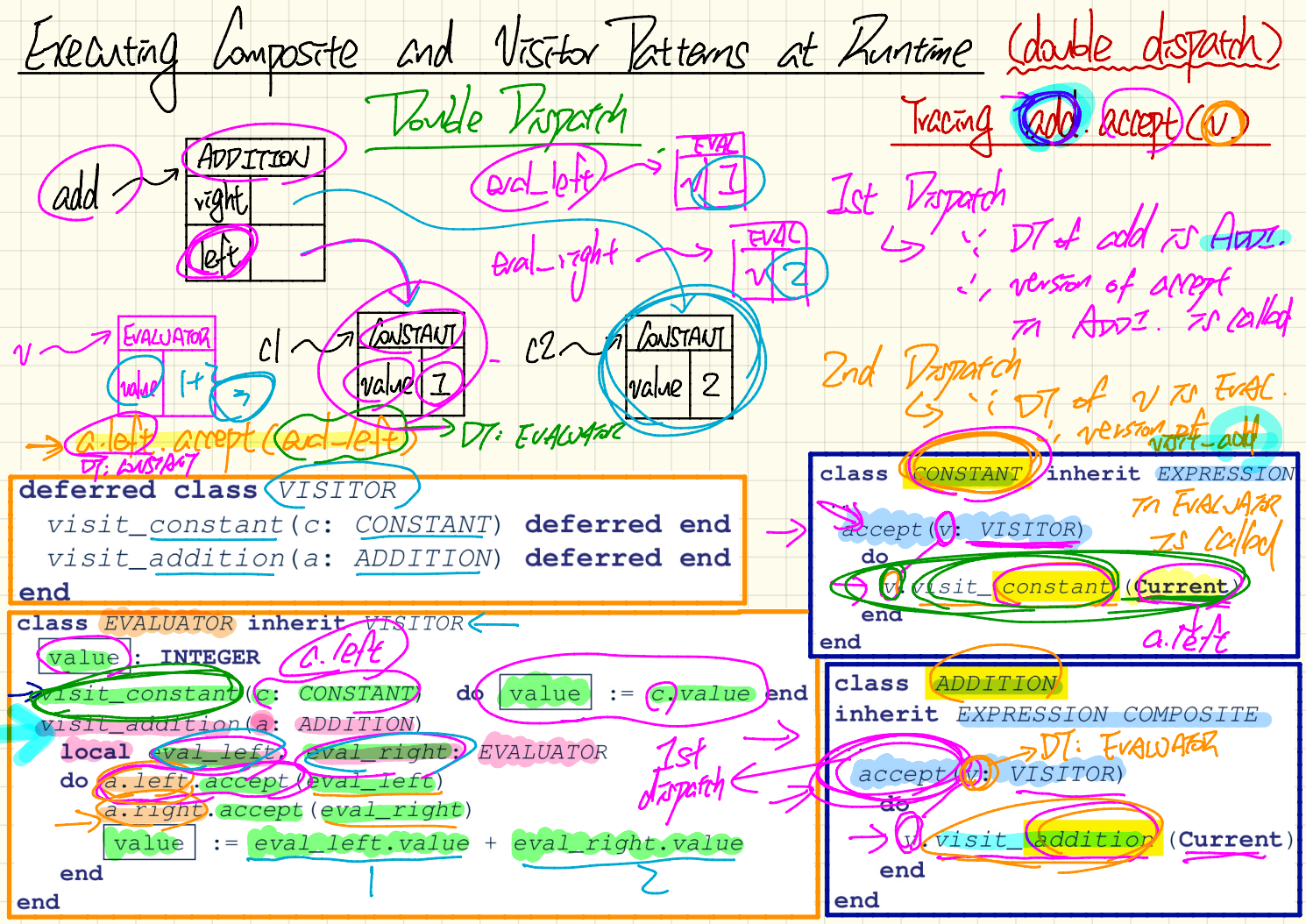
Visualizing Line 4 to Line 7



1st dispatch : send add

2nd dispatch : EVALUATOR vs. TYPE_CHECKER

Executing Composite and Visitor Patterns at Runtime (double dispatch)



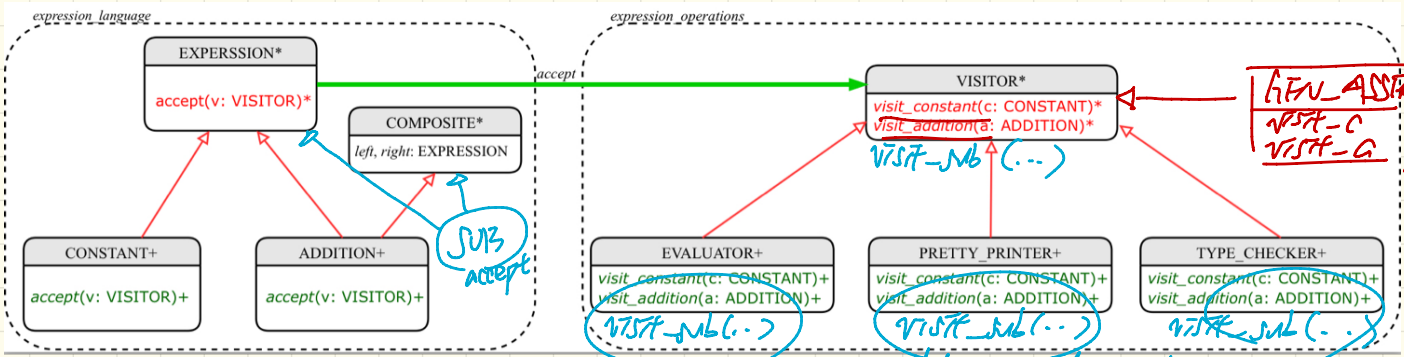

```
deferred class VISITOR
  visit_constant(c: CONSTANT) deferred end
  visit_addition(a: ADDITION) deferred end
end
```

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

```
class EVALUATOR inherit VISITOR
  value: INTEGER
  visit_constant(c: CONSTANT) do value := c.value end
  visit_addition(a: ADDITION)
    local eval_left, eval_right: EVALUATOR
    do a.left.accept(eval_left)
      a.right.accept(eval_right)
      value := eval_left.value + eval_right.value
    end
end
```

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

Visitor Pattern: Open-Closed and Single Choice Principles



Adding a new language construct?

↳ not good :(this is supposed to be closed for VISITOR.

① add a descendant to EXP.

② change every descendant of VISITOR

↳ violate SCP

Adding a new language operation?

✓

① add a descendant to VISITOR