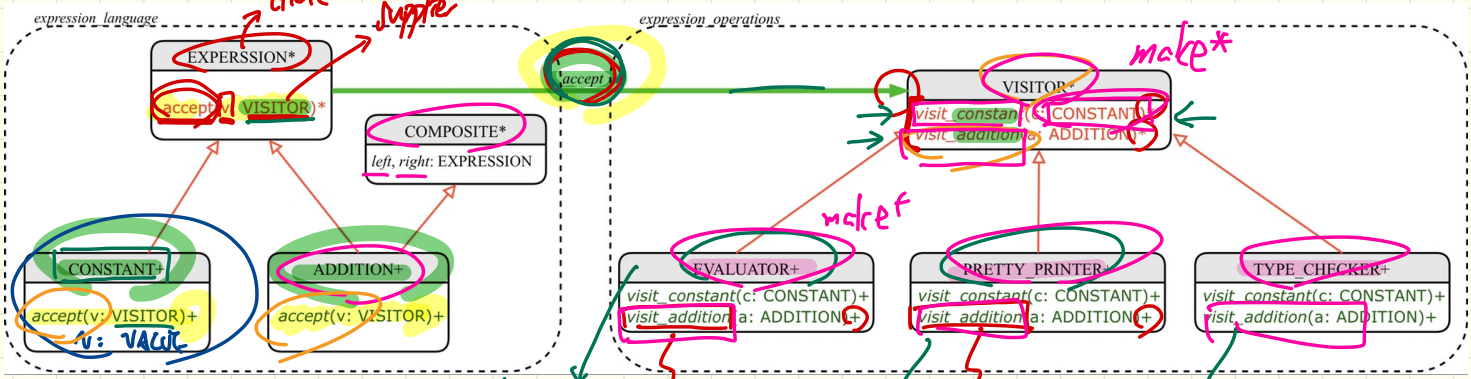
Lecture 15
Thursday October 31

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
```

add. accept (v)

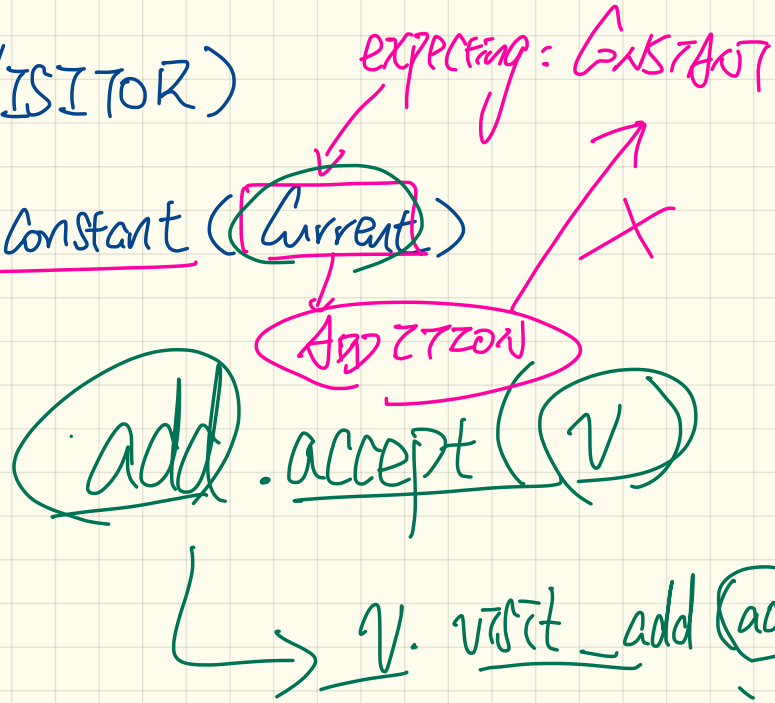
root of expression tree

operation to perform

class ADDITION

inherit EXPRESSION

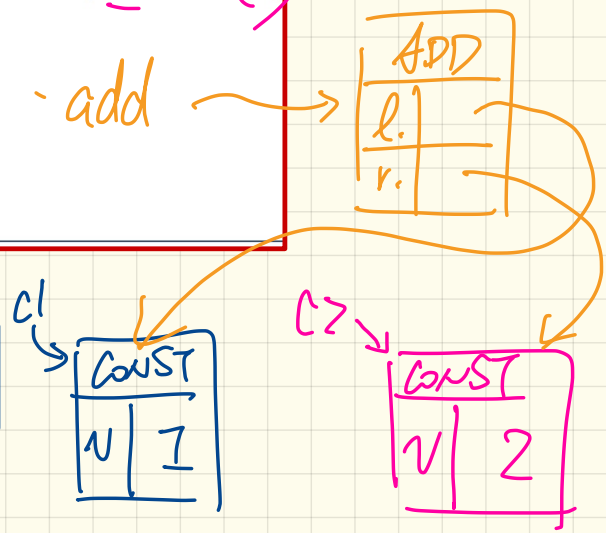
```
accept (v: VISITOR)
    do
        v.visit_constant (Current)
    end
```



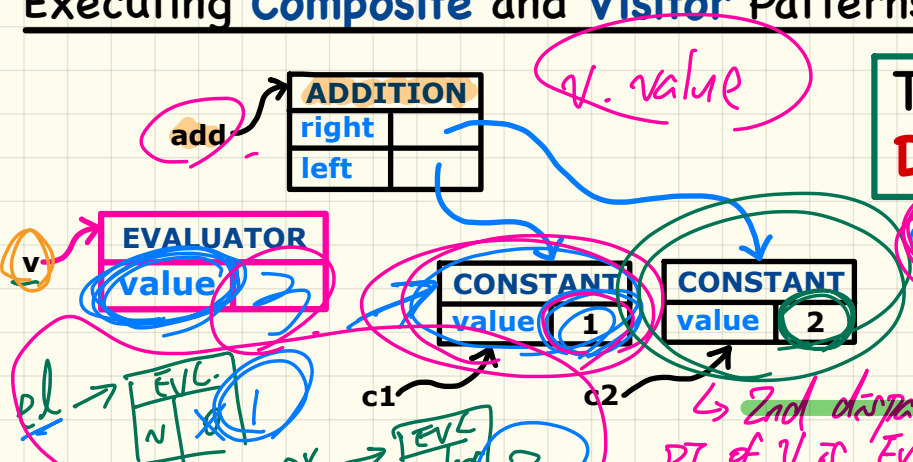
Visitor Design Pattern: Implementation

```
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
```

Visualizing Line 4 to Line 6



Executing Composite and Visitor Patterns at Runtime



Tracing add.accept(v)
Double Dispatch

add.accept(v)

1st dispatch:
DT of add is ADDI.
∴ call accept from ADDI.

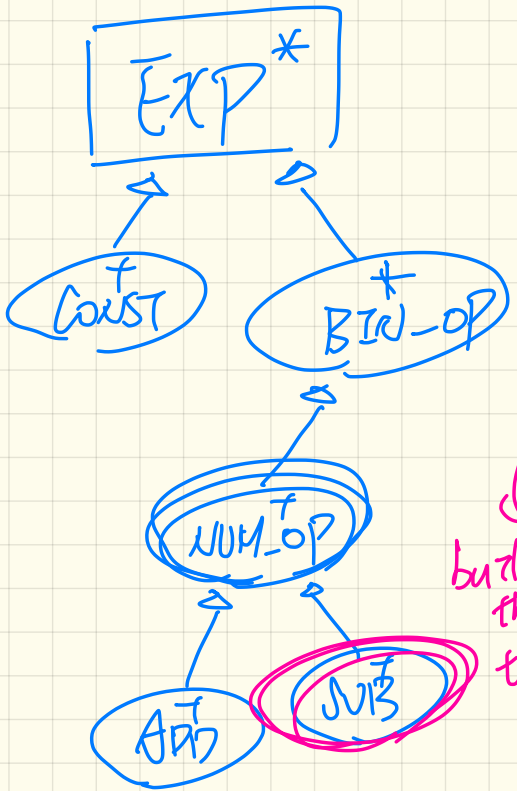
```
deferred class VISITOR
  visit_constant(c: CONSTANT) deferred end
  visit_addition(a: ADDITION) deferred end
end
```

```
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
```

Exercise: Explain the D.D. here.

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



eval: EVALUATOR (2-3)-(4-7)

e : EXP.

create {SUB} e.make

build the tree

create eval.make

which tells which version of visit_sub feature to call.

e.accept(eval)

↳ 1st: DT of e: SUB

2nd: DT of eval is EVAL

v: VISITOR

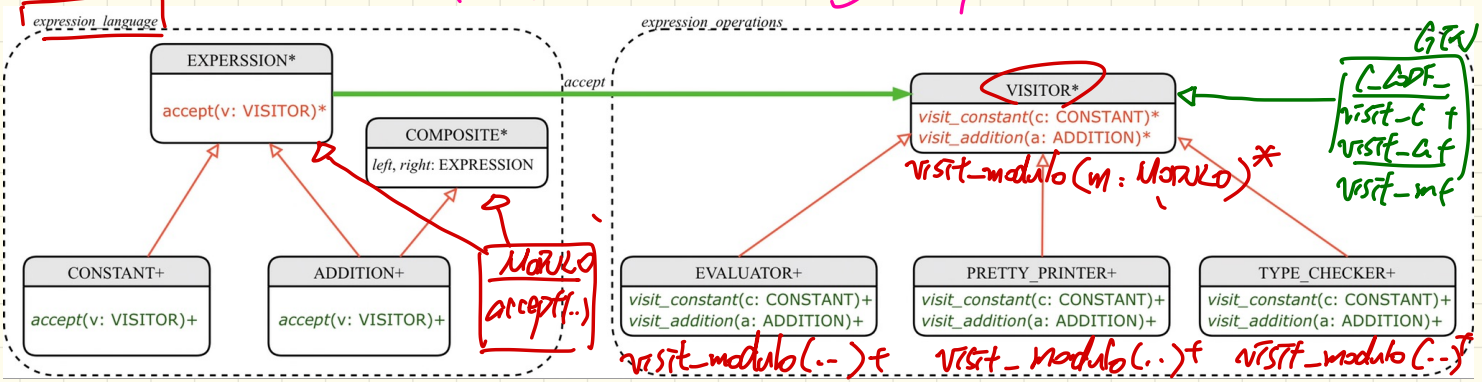
{ }

create v. make

↳ create {VISITOR} v. make ✗

*

Visitor Pattern: Open-Closed and Single-Choice Principles



What if a new language construct is added?

What if a new language operation is added?

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

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