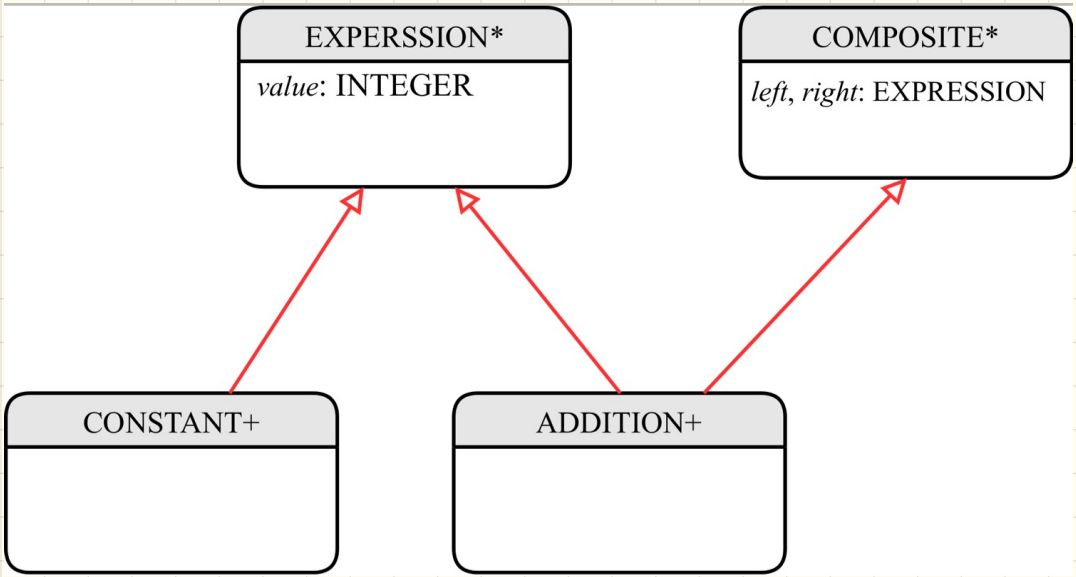
Lecture 11

Part 1

Processing Recursive Systems

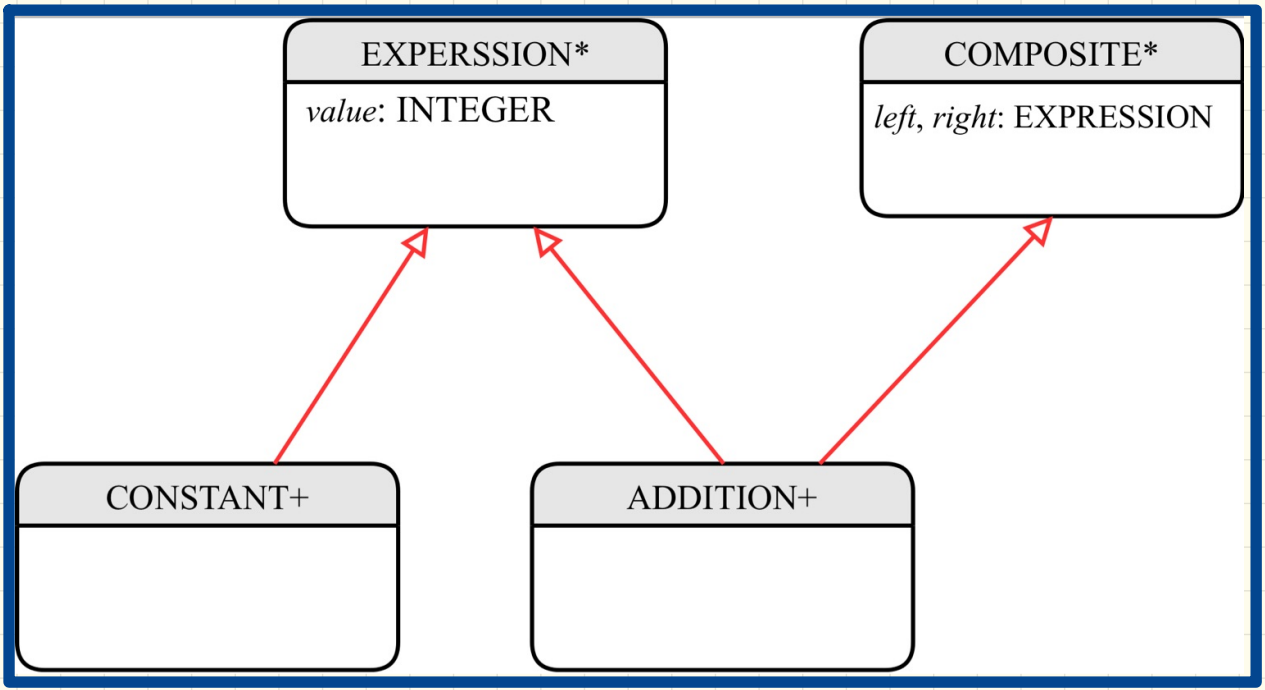
Design of Language Structure: Composite Pattern



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

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

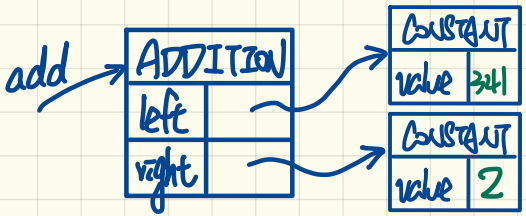
Design of Language Operation: How to Extend the Composite Pattern?



Structure

- evaluate
- print_prefix
- print_postfix
- type_check

Operations

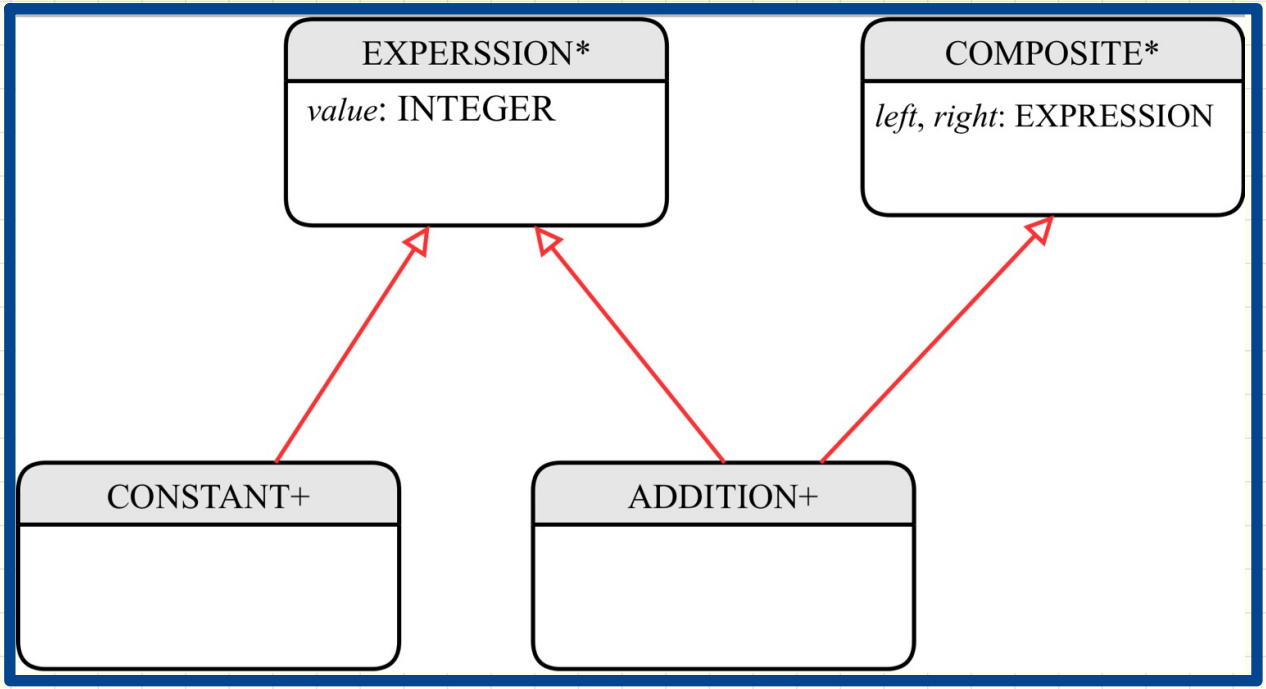


Lecture 11

Part 2

Open-Closed Principle

Design of a Language Application: Open-Closed Principle



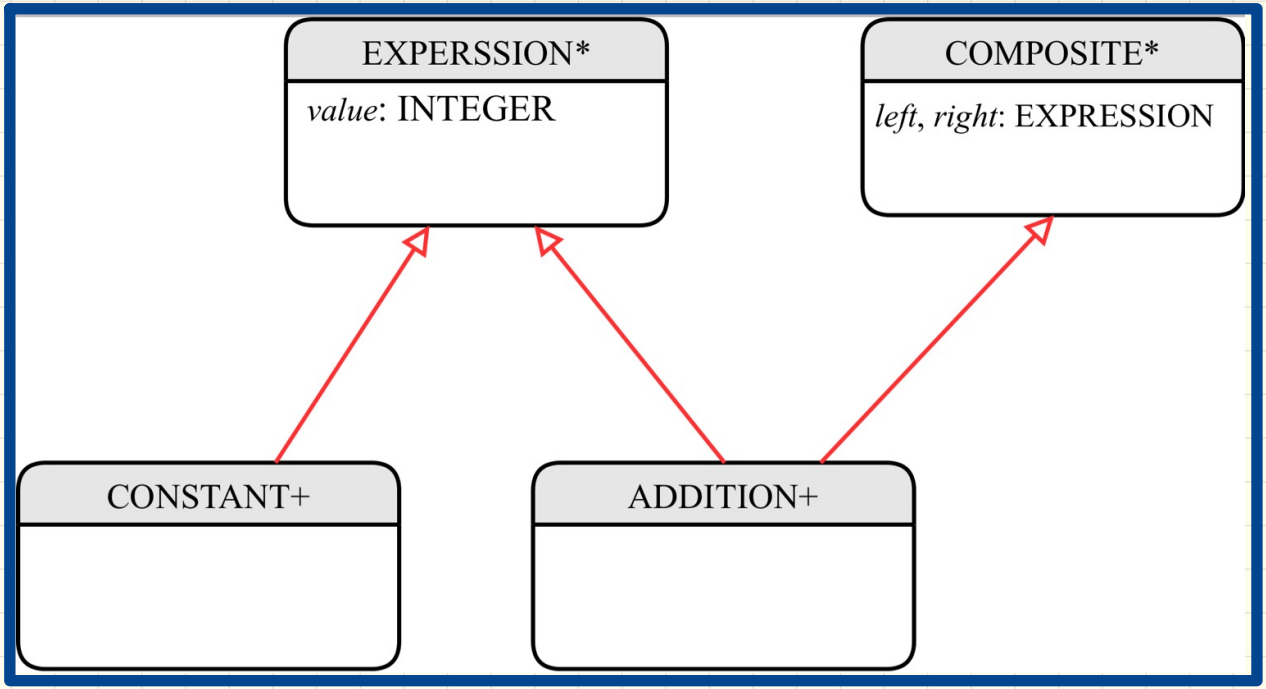
Structure

evaluate
print_prefix
print_postfix
type_check

Operations

	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

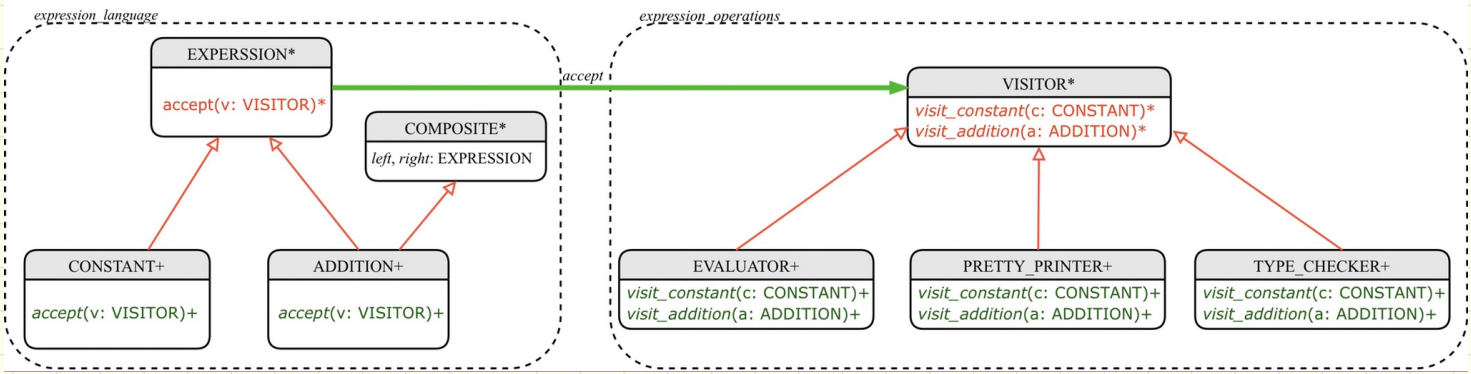
	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Lecture 11

Part 3

Visitor Design Pattern

Visitor Design Pattern: Architecture



How to Use Visitors

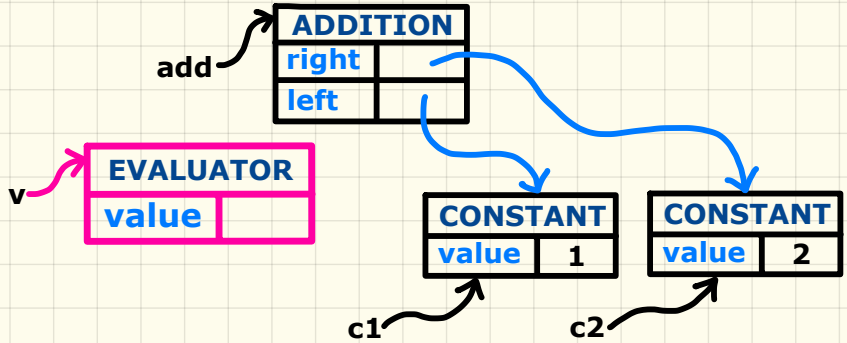
```
1  test_expression_evaluation: BOOLEAN
2   local add, c1, c2: EXPRESSION ; v: VISITOR
3   do
4     create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
5     create {ADDITION} add.make (c1, c2)
6     create {EVALUATOR} v.make
7     add.accept (v)
8     check attached {EVALUATOR} v as eval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

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

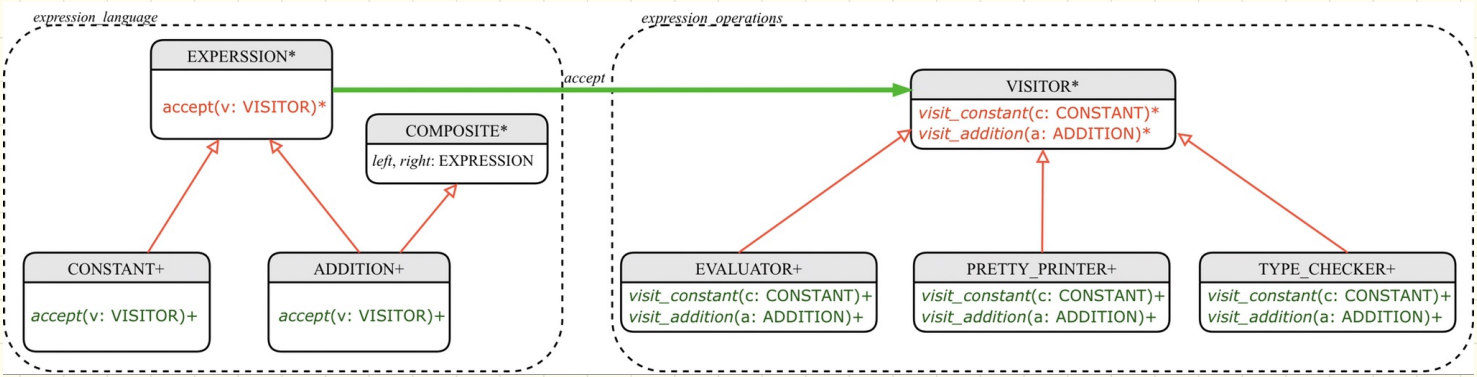
```
deferred class VISITOR
  visit_constant(c: CONSTANT) deferred end
  visit_addition(a: ADDITION) deferred end
end
```

```
class EVALUATOR inherit VISITOR
  value: INTEGER
  visit_constant(c: CONSTANT)  do value := c.value end
  visit_addition(a: ADDITION)
    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 CONSTANT inherit EXPRESSION
...
  accept(v: VISITOR)
    do
      v.visit_constant(Current)
    end
end
```

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

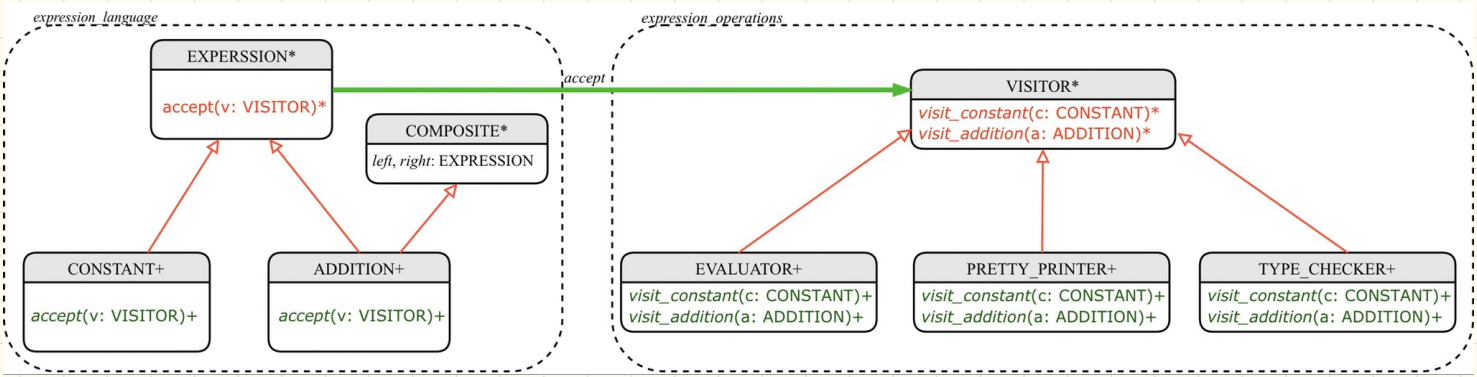
Visitor Pattern: Open-Closed and Single-Choice Principles



What if a new language construct is added?

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

Visitor Pattern: Open-Closed and Single-Choice Principles



What if a new language operation is added?

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