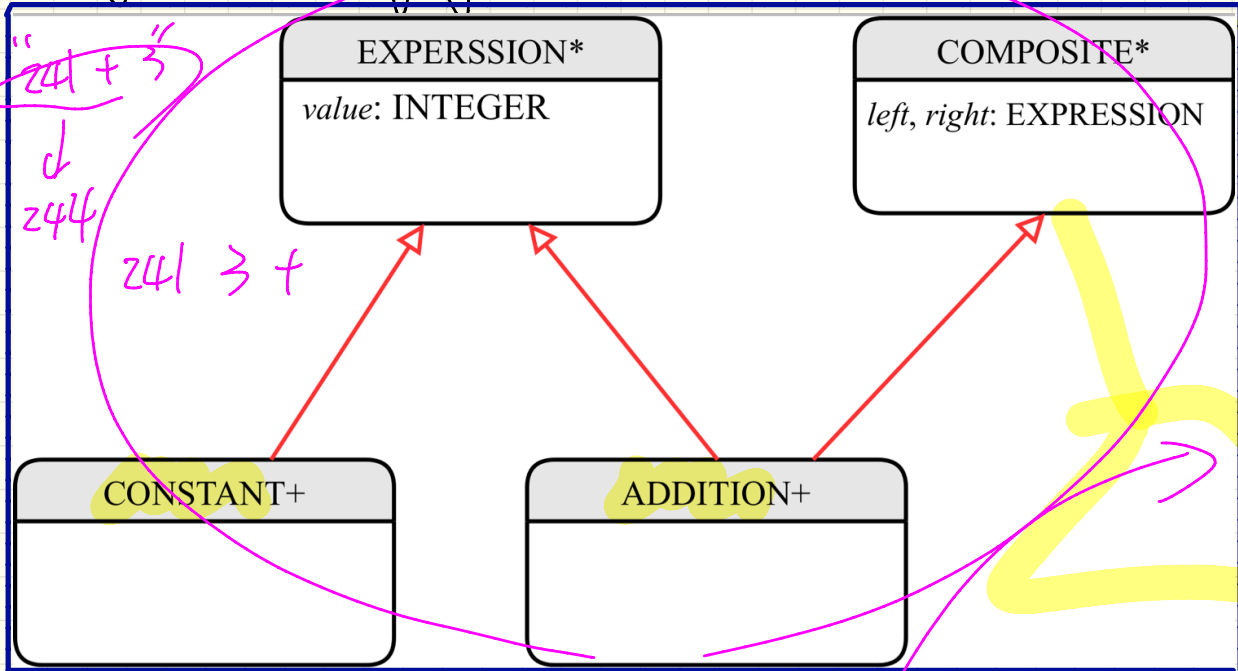


Tuesday Nov. 6
Lecture 16

- Exam: Sunday Dec. 9 7pm
- Midterm results available this Thursday
- Lab → (programming) marks available early Friday
- Project released by next Wed.
(→ weeks)

Design of a Language Application: Open-Closed Principle



Structure

"244 + 3"
↓
244

244 | 3 | +

app. context of visitor

Operations:

- evaluate
- print - prefix
- print - postfix
- type - check

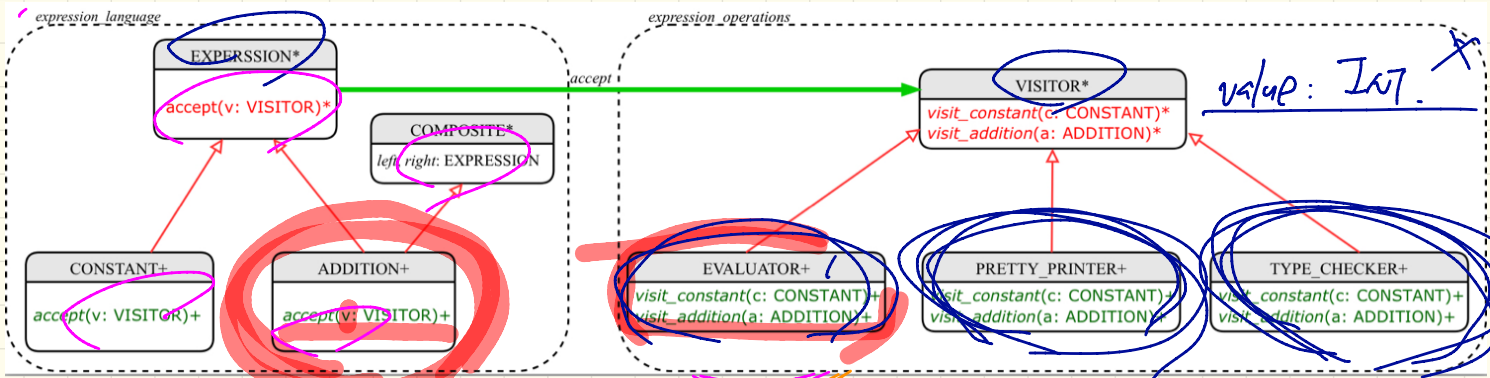
Operations
gen. - java code

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

Visitor Design Pattern: Architecture

add accept

234 + "a"



How to Use Visitors

effective descendants of EXPRESSION
 add.accept(v) ⇒ create a visit_x pattern
 c1.accept(v) ⇒ in VISITOR

```

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
    
```

c2.accept(v) → 2

234 + 2 = 236
 eval.visit_add(c_e)
 pp.visit_add(c_e)
 "234 2 +"

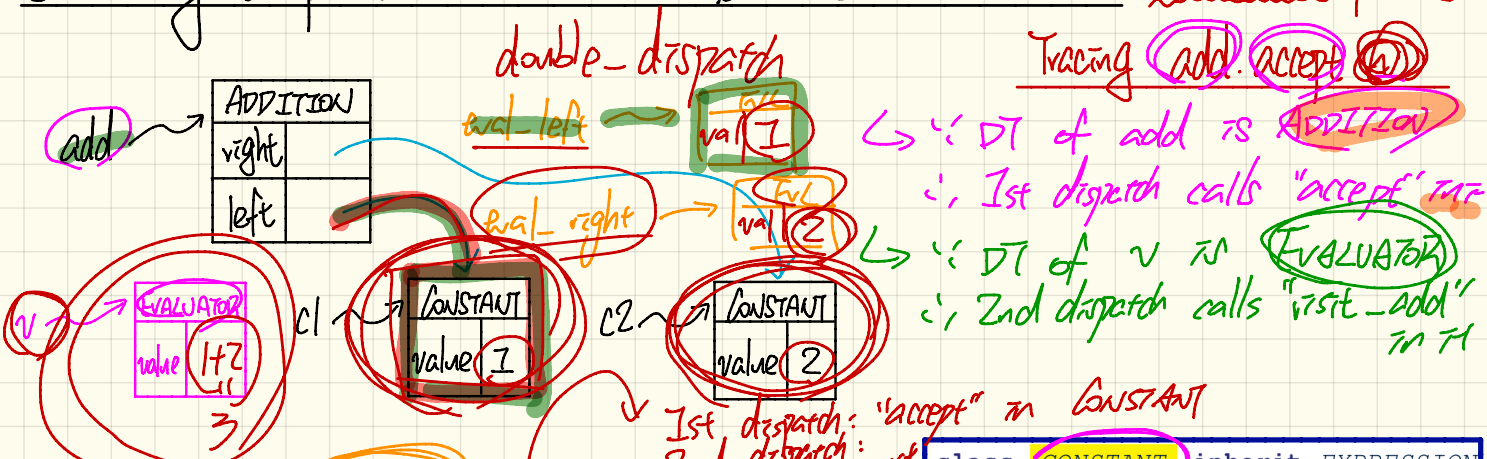
Visitor Design Pattern: Implementation

```
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

$$1 + 2$$

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

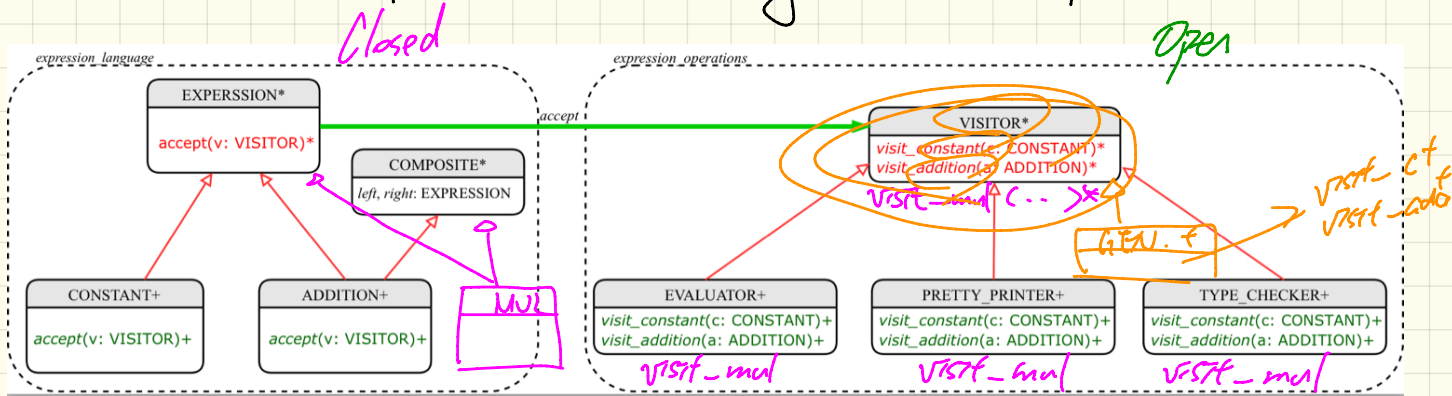
1st dispatch: "accept" in CONSTANT
 2nd dispatch: "visit_constant" in EVALUATOR

Handwritten annotations on the VISITOR code:
 - `visit_addition` is circled in green.
 - `do a.left.accept(eval_left)` and `do a.right.accept(eval_right)` are circled in red.
 - `value := eval_left.value + eval_right.value` is circled in red.
 - `a.left.accept(eval_left)` is circled in red.

Handwritten annotations on the CONSTANT code:
 - `accept(v: VISITOR)` is circled in red.
 - `do v.visit_constant(Current)` is circled in red.
 - `eval_left DT: EVALUATOR` is written in red above the code.

Handwritten annotations on the ADDITION code:
 - `accept(v: VISITOR)` is circled in red.
 - `do v.visit_addition(Current)` is circled in red.
 - `DT: EVALUATOR` is written in red above the code.

Visitor Pattern: Open-Closed and Single Choice Principles



Adding a new language construct? →

violates SCP
 ⇒ this part should be closed

Adding a new language operation? →

satisfied SCP
 ⇒ this part can be open

GEN-ASSEMBLY

Void Safe in Java? (1)

```
1 class Point {
2   double x;
3   double y;
4   Point(double x, double y) {
5     this.x = x;
6     this.y = y;
7   }
```

```
1 class PointCollector {
2   ArrayList<Point> points;
3   PointCollector() { }
4   void addPoint(Point p) {
5     Null | points.add(p); }
6   Point getPointAt(int i) {
7     return points.get(i); }
```

The above Java code **compiles**. But anything wrong?

```
1 @Test
2 public void test1() {
3   → PointCollector pc = new PointCollector();
4   → pc.addPoint(new Point(3, 4));
5   Point p = pc.getPointAt(0);
6   assertTrue(p.x == 3 && p.y == 4); }
```

pc.points null

Void Safe in Java? (2)

```
1 class Point {
2     double x;
3     double y;
4     Point(double x, double y) {
5         this.x = x;
6         this.y = y;
7     }
}
```

```
1 class PointCollector {
2     ArrayList<Point> points;
3     PointCollector() {
4         points = new ArrayList<>();
5     }
6     void addPoint(Point p) {
7         points.add(p);
8     }
9     Point getPointAt(int i) {
10        return points.get(i);
11    }
}
```

```
1 @Test
2 public void test2() {
3     PointCollector pc = new PointCollector();
4     Point p = null;
5     pc.addPoint(p);
6     p = pc.getPointAt(0);
7     assertTrue(p.x == 3 && p.y == 4);
}
```

Handwritten annotations: "null" with arrows pointing to the `p` variable in line 4 and line 6. A "null" with an arrow pointing to the `new` keyword in line 3. A "null" with an arrow pointing to the `getPointAt(0)` call in line 6.