Wednesday April 10

Review Lecture

```
expanded class UTIL
  is_positive (i: INTEGER) : BOOLEAN
    do
      Result := i > 0
    end

  counting (a: ARRAY[INT]; f: FUNCTION[INT, BOOL]) : INT
    do
      across a as cursor loop
        if f (a.item) then Result :=
                               Result+1 end
      end
    end
```

cursor

param

return type

1	2	3	-1
✓	✓	✓	X

agent u.is_positive(2)

agent u.is_positive(-2)

```
test: BOOL
  local
    a: ARRAY[INT]
    u: UTIL
  do
    a := << 1, 2, 3, -1 >>
    Result := u.counting(a, agent u.is_positive(?) )
  end
```

3

0??

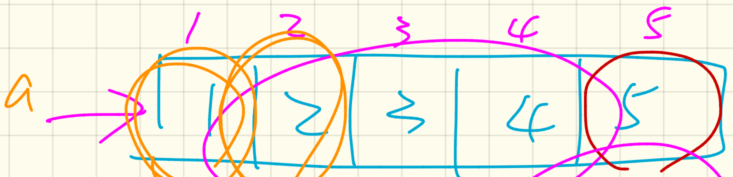
= ?? 4

expanded

agent & add

```
class UTIL
  add (i, j: INT): INT
    do
      Result := i + j
    end

  counting (a: A[INT];
            f: FUNCTION[INT, INT, INT]) : INT
    require
      a.count >= 2
    local
      i: INT
    do
      from i := a.lower
      until i = a.upper - 1
      loop
        Result := Result + f(a[i], a[i+1])
        i := i + 1
      end
```



1 + 2

2 + 3

3 + 4

4 + 5

5 7 9

agent u.add (?, ?)

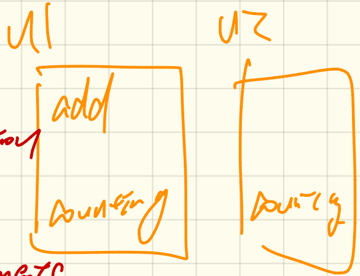
[cursor.item,
(cursor + 1).item]

test: Bool

① pattern of loop

② signature of function

③ agent and open arguments



```
local
  a: A[I]
  u1: UTIL
  u2
do
  a := <<1, 2, 3, 4, 5>>
  Result := u.counting (a, agent u.add (?, ?))
end
```

" 24.

FUNCTION[INT, INT, INT]

add(2, 3) → 5

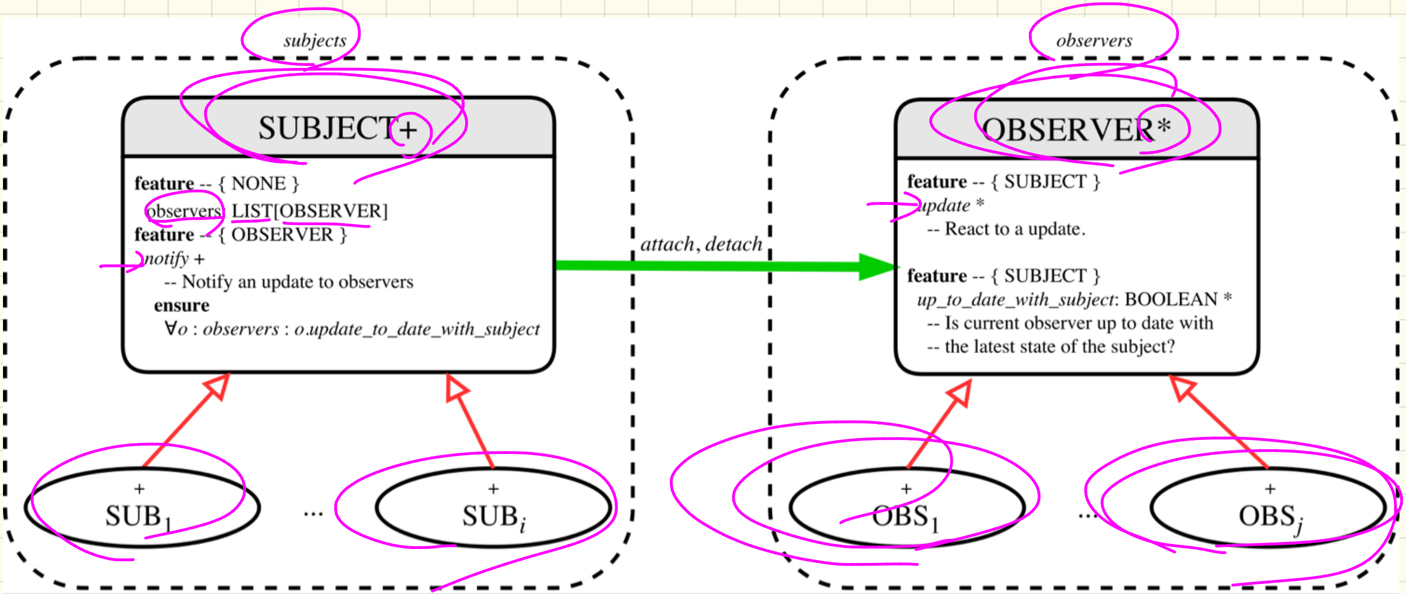
PROCEDURE[INT]

increment_by(3)

^ function returning boolean

PREDICATE[INT] → is_positive(3)

The Observer Pattern



subjects

SUBJECT+

feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
∀*o* : *observers* : *o.update_to_date_with_subject*

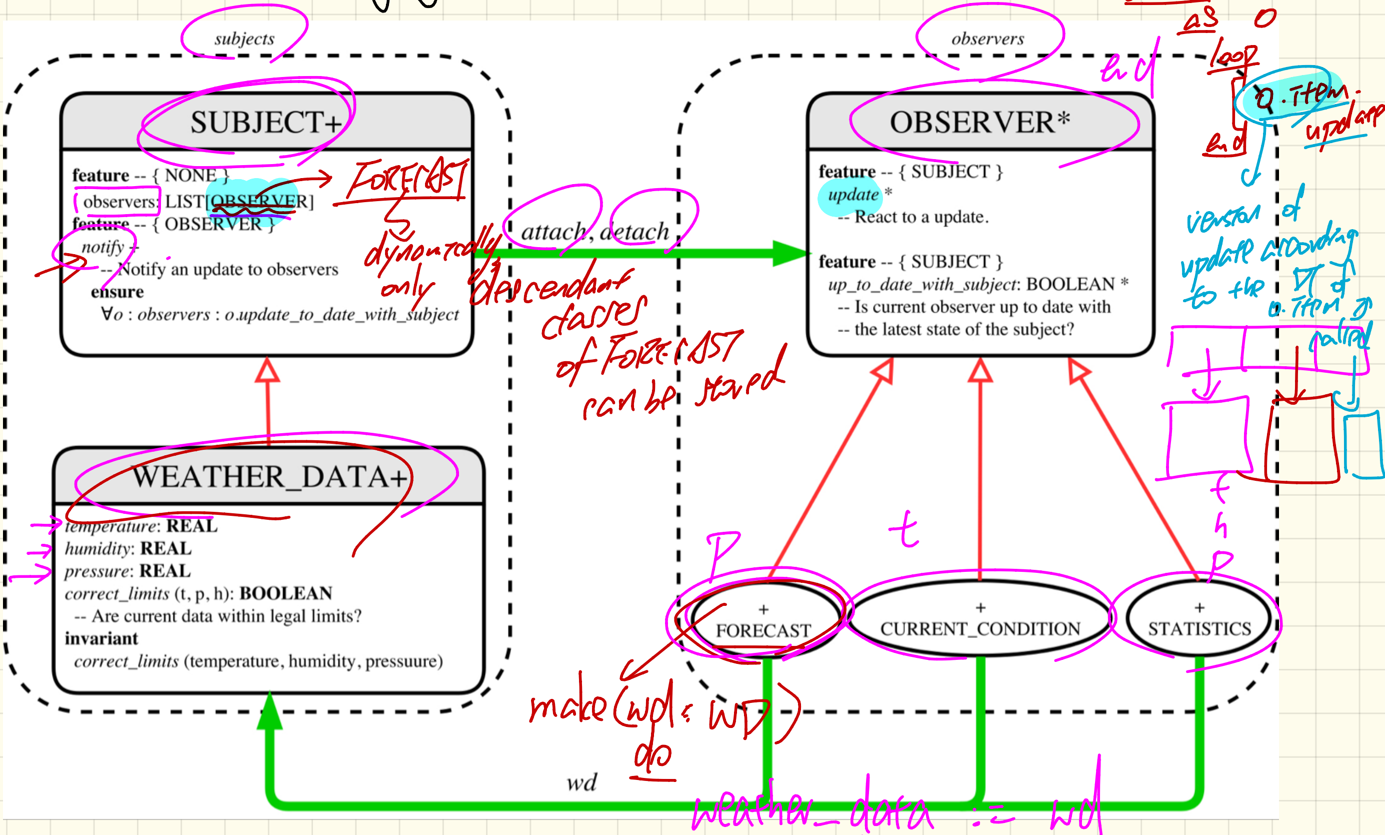
observers

OBSERVER*

feature -- { SUBJECT }
update *
-- React to a update.

feature -- { SUBJECT }
up_to_date_with_subject: BOOLEAN *
-- Is current observer up to date with
-- the latest state of the subject?

Weather Station: Applying the Observer Pattern



subjects

SUBJECT+

feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
∀*o* : *observers* : *o.update_to_date_with_subject*

WEATHER_DATA+

temperature: **REAL**
humidity: **REAL**
pressure: **REAL**
correct_limits (t, p, h): **BOOLEAN**
-- Are current data within legal limits?
invariant
correct_limits (temperature, humidity, pressuure)

attach, *detach*

observers

OBSERVER*

feature -- { SUBJECT }
update *
-- React to a update.

feature -- { SUBJECT }
up_to_date_with_subject: BOOLEAN *
-- Is current observer up to date with
-- the latest state of the subject?

+ FORECAST

+ CURRENT_CONDITION

+ STATISTICS

wd

FORECAST

dynamically only descendant classes of FORECAST can be stored

p

t

h

p

```
notify
do
  across observers as o
  loop
    o.item.update
  end
end
```

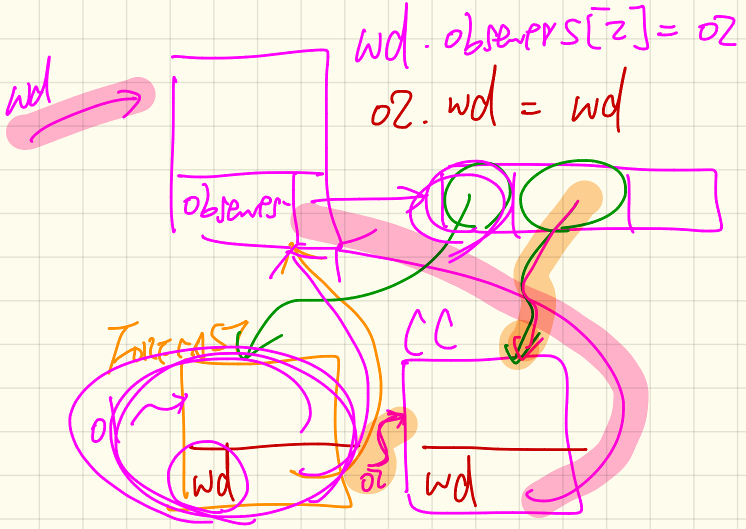
version of update according to the DT of o.item

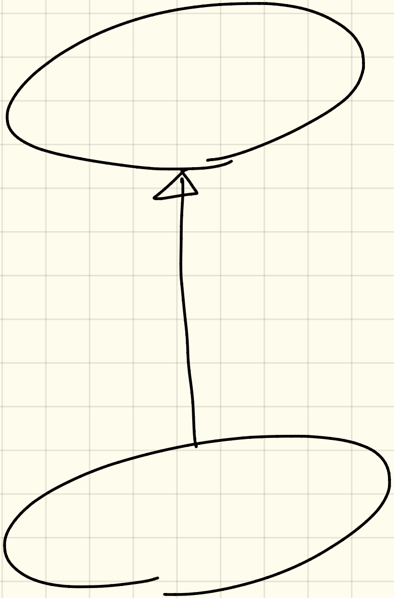
```
make(wd: WD)
do
  weather_data := wd
  wd.attach(Current)
end
```

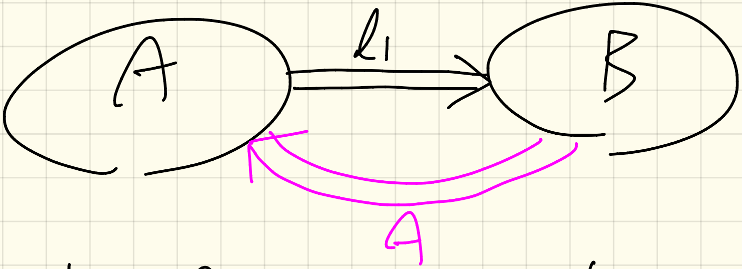
Implementing Weather Station: Subject

```
class SUBJECT create make
feature -- Attributes
  observers : LIST[OBSERVER]
feature -- Commands
  make
   do create {LINKED_LIST[OBSERVER]} observers.make
   ensure no_observers:  observers.count = 0 end
feature -- Invoked by an OBSERVER
  attach (o: OBSERVER) -- Add ’o’ to the observers
    require not_yet_attached: not observers.has (o)
    ensure is_attached: observers.has (o) end
  detach (o: OBSERVER) -- Add ’o’ to the observers
    require currently_attached: observers.has (o)
    ensure is_attached: not observers.has (o) end
feature -- invoked by a SUBJECT
  notify -- Notify each attached observer about the update.
    do across observers as cursor loop cursor.item.update end
    ensure all_views_updated:
      across observers as o all o.item.up_to_date_with_subject end
    end
end
```

```
class WEATHER_DATA
inherit SUBJECT  rename make as make_subject end
create make
feature -- data available to observers
  temperature: REAL
  humidity: REAL
  pressure: REAL
  correct_limits(t,p,h: REAL): BOOLEAN
feature -- Initialization
  make (t, p, h: REAL)
    do
      make_subject -- initialize empty observers
      set_measurements (t, p, h)
    end
feature -- Called by weather station
  set_measurements(t, p, h: REAL)
    require correct_limits(t,p,h)
invariant
  correct_limits(temperature, pressure, humidity)
end
```





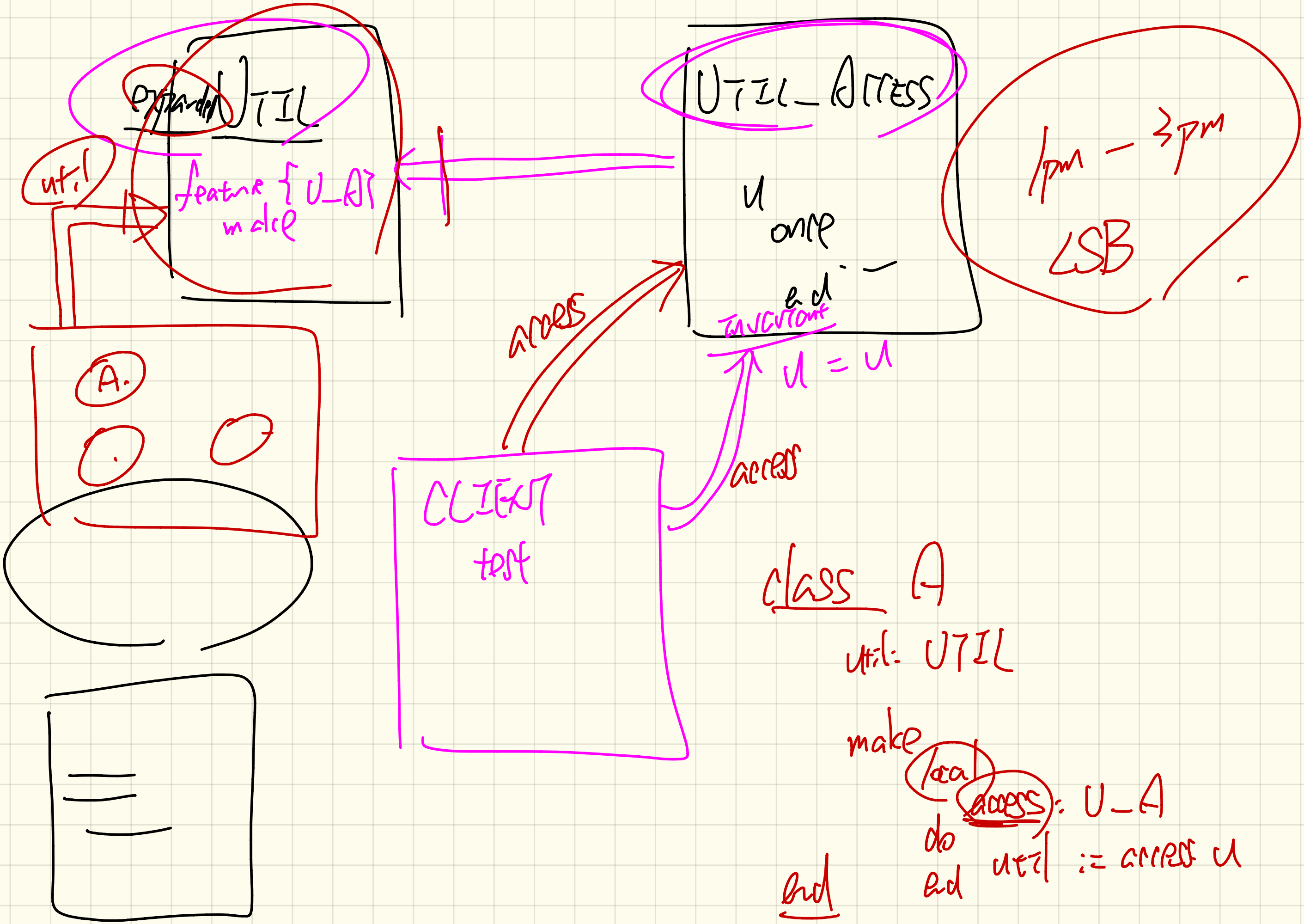


class A

l1 : B

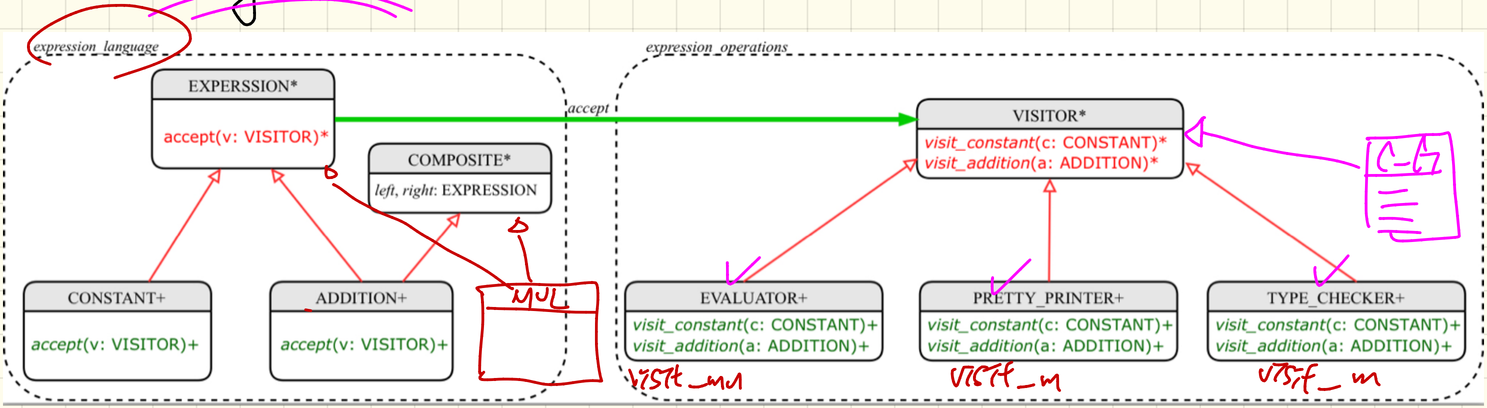
class B

l2 : A



```
class A
    util: UTIL
    make
        local
            access: U_A
        do
            util := access u
        end
end
```

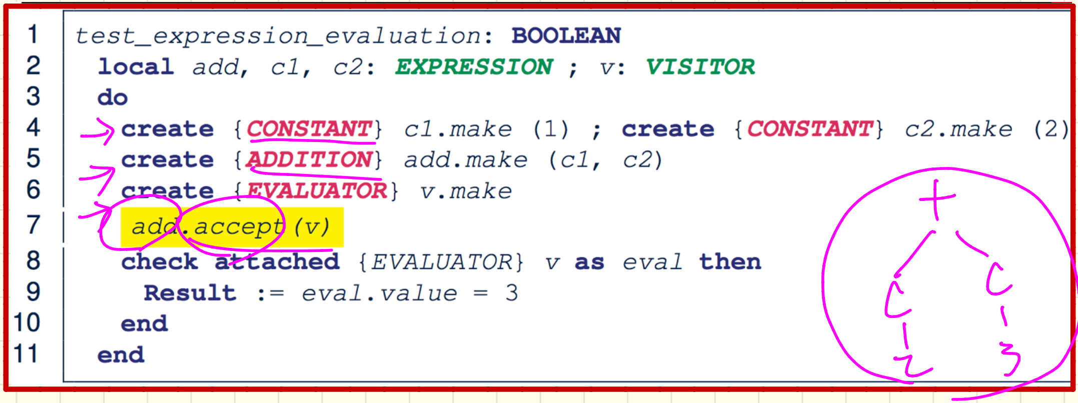
Visitor Design Pattern: Architecture



OCP → closed
open

change 2: add a new operation

How to Use Visitors



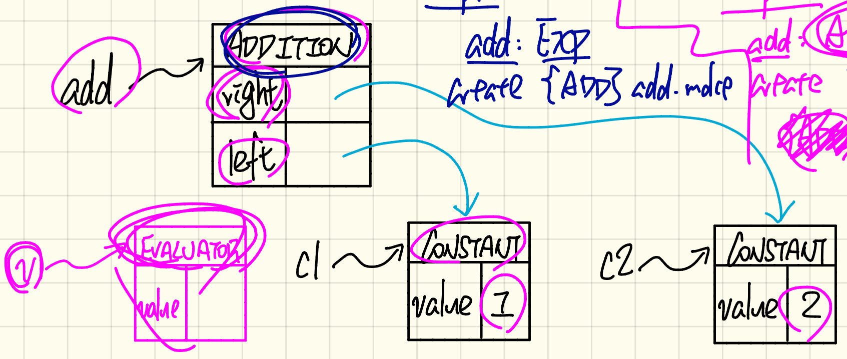
```
test_expression_evaluation: BOOLEAN
 local add, c1, c2: EXPRESSION ; v: VISITOR
 do
   create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
   create {ADDITION} add.make (c1, c2)
   create {EVALUATOR} v.make
   add.accept (v)
   check attached {EVALUATOR} v as eval then
     Result := eval.value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

change 1: add a new structural comp.

MULTIPLICATION

SCP.

Executing Composite and Visitor Patterns at Runtime (double dispatch)



Opt 1

add: EXP

create {ADD} add.make

Opt 2

add: ADDITION

create add.make

Tracing add.accept(v)

1st Dispatch: DT of add is ADDITION; call the accept in

2nd Dispatch DT of v is EVALUATOR, call the visit_add

```
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
```

in EVALUATOR

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