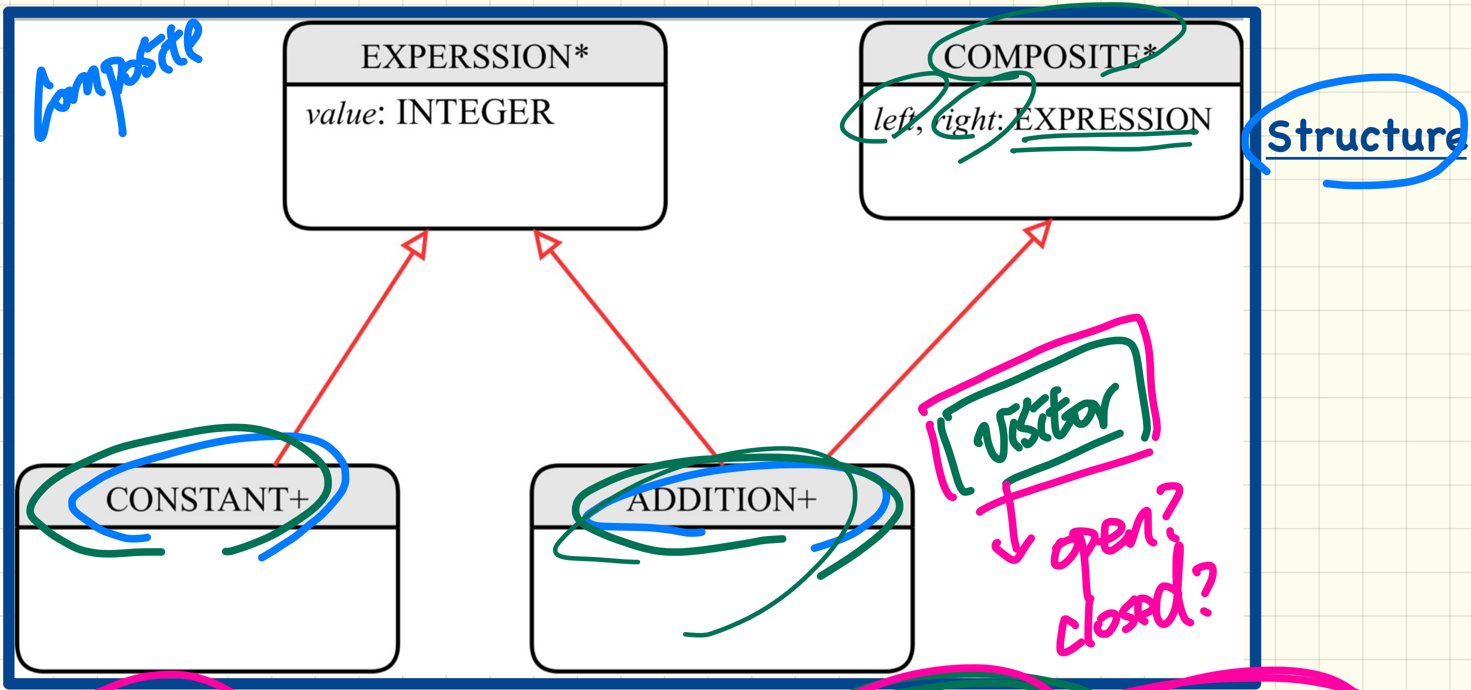
LECTURE 22

WEDNESDAY MARCH 25

Design of a Language Application: Open-Closed Principle



Composite

EXPERSSION*

value: INTEGER

COMPOSITE*

left, *right*: EXPRESSION

Structure

CONSTANT+

ADDITION+

Visitor

open?

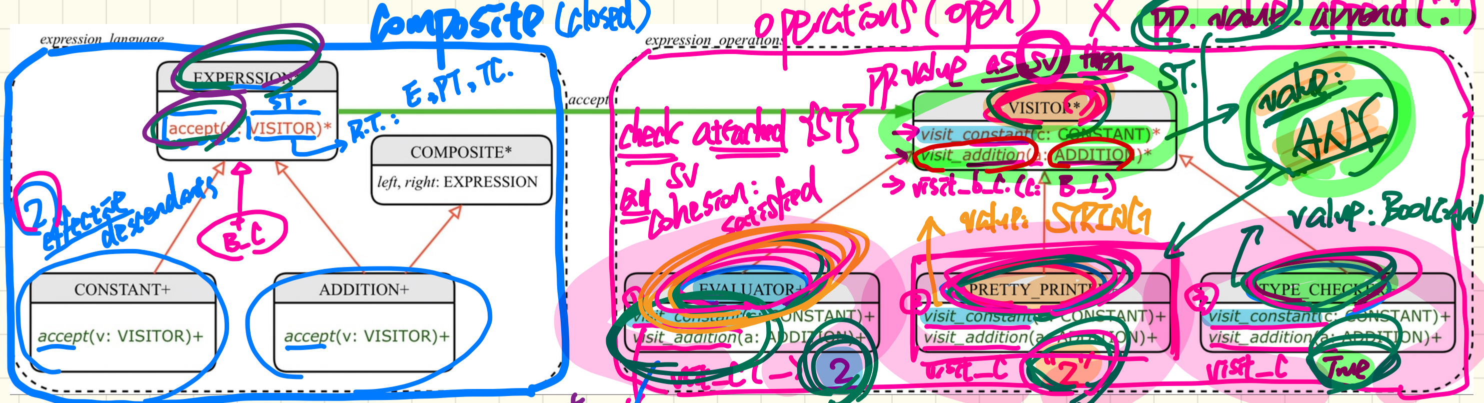
closed?

evaluate
print_prefix
print_postfix
type_check

Operations

	Structure	Operations
Alternative 1	Open	Closed
Alternative 2	Closed	Open

Visitor Design Pattern: Architecture

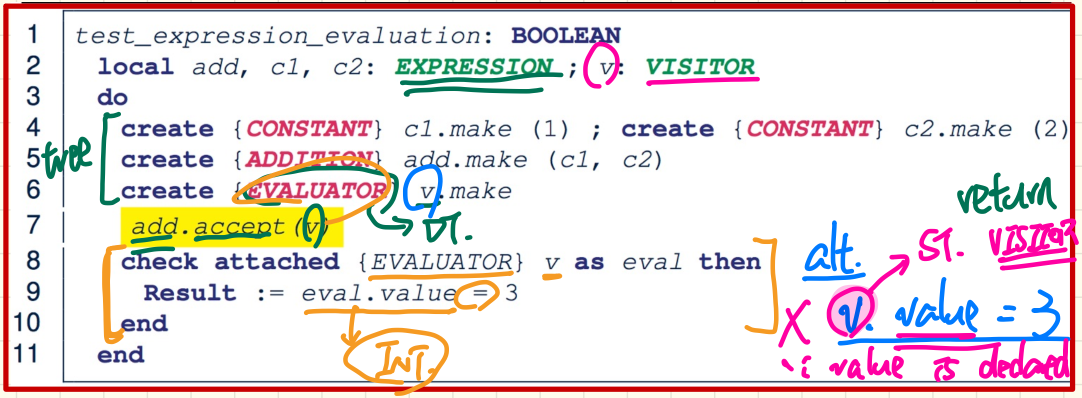


v. — e.accept

value: INT

①, or ②, or ③ which version to call at R.T depends on the D.T. of visitor

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
```

For each descendant of VISITOR, the return value may be of different type.

at the individual value visitor descend.

Eiffel

(no feature overloading)

check attached

if attached

visit_constant (CONSTANT)

visit_addition (ADDITION)

Java

(supports method overloading)

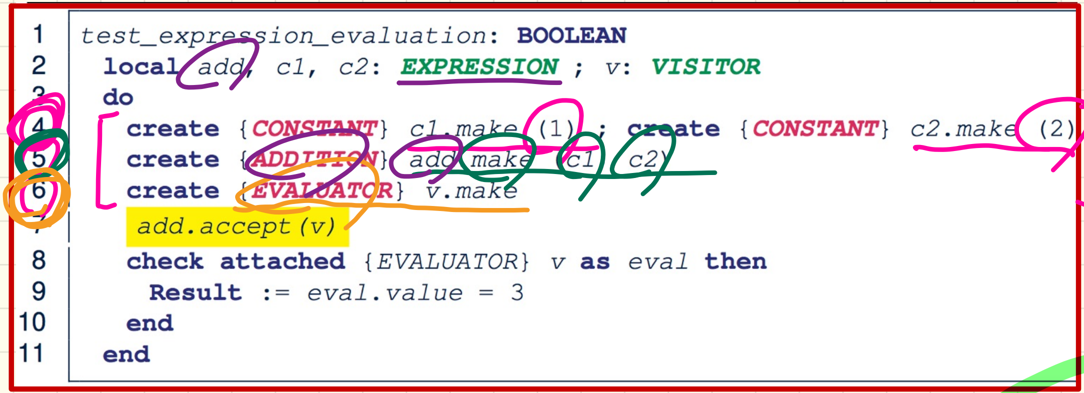
① visitConstant(Constant)

visitAddition(Addition)

② visit (Constant c)

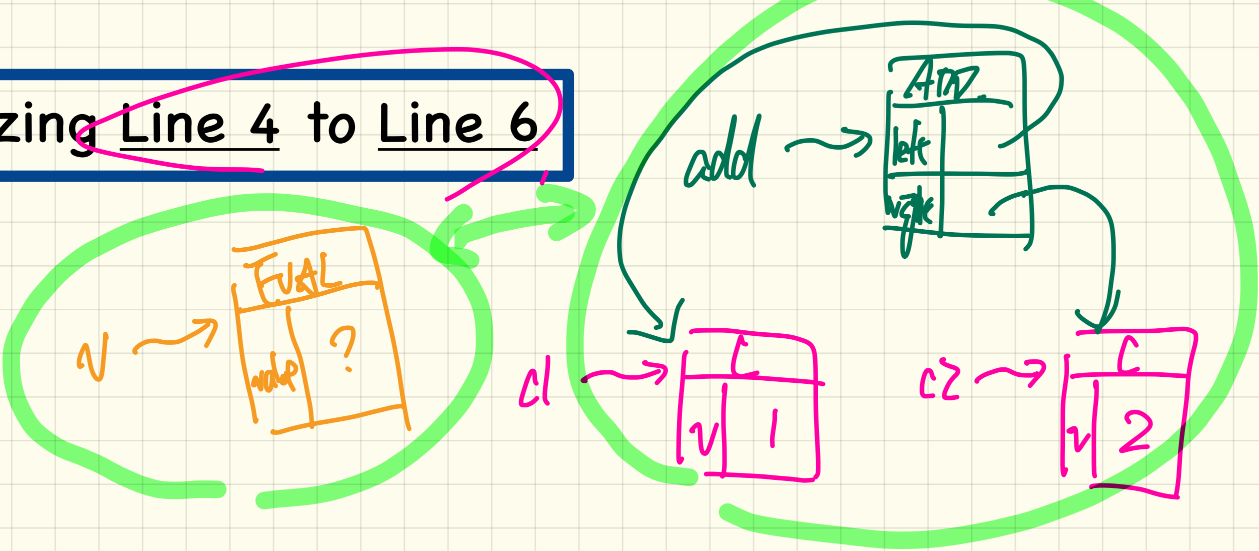
visit (Addition a)

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



eval_left. value + eval_right. value

Evaluate addition

1 + 2

1 2

(1+2) + (3+4)

How do we evaluate left & right recursively, then combine their results using "+"?

add

eval : EVALUATOR

eval. visit_addition (add)

eval_right : EVALUATOR.

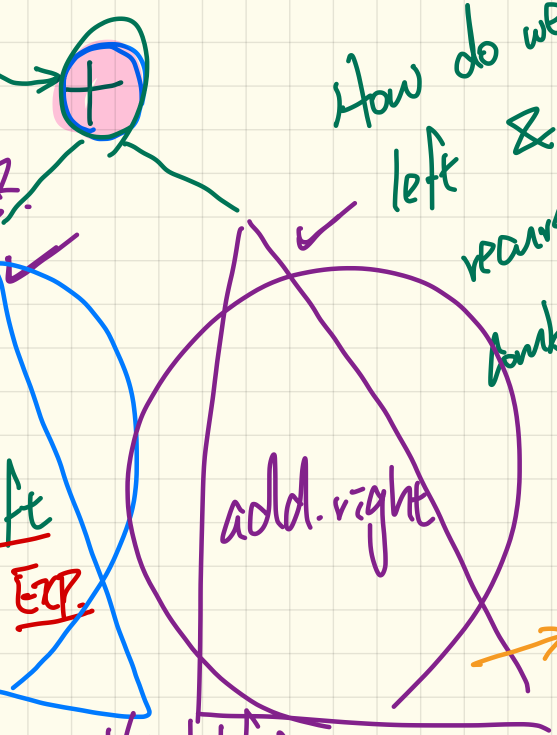
eval_left :

~~visit_addition(~~ ADD.

eval_left. visit_addition (add.left)

~~ADDITION~~ X → not compile

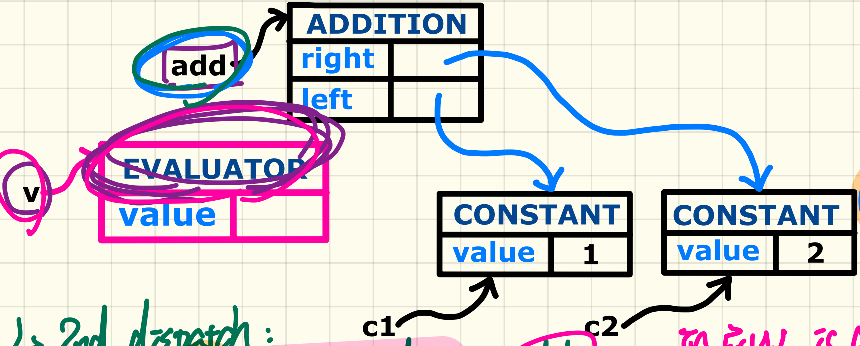
eval_left. value



add. left. accept(eval_left)

→ add.right. accept(eval_r.)

Executing Composite and Visitor Patterns at Runtime



Tracing add.accept(v)

Double Dispatch

↳ dyn. bind.

× 2

add.accept(v)

↳ 1st dispatch:
∵ DT of add is ADDITION
∴ accept in ADDITION is called
↳ execute: v. visit_add (add)

↳ 2nd dispatch:
v. visit_addition (add)
∵ DT of v is EVAL. ∴ visit_add in EVAL is called

```
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
```

add

add
add

Exercise. Explain D.D. for ____ .

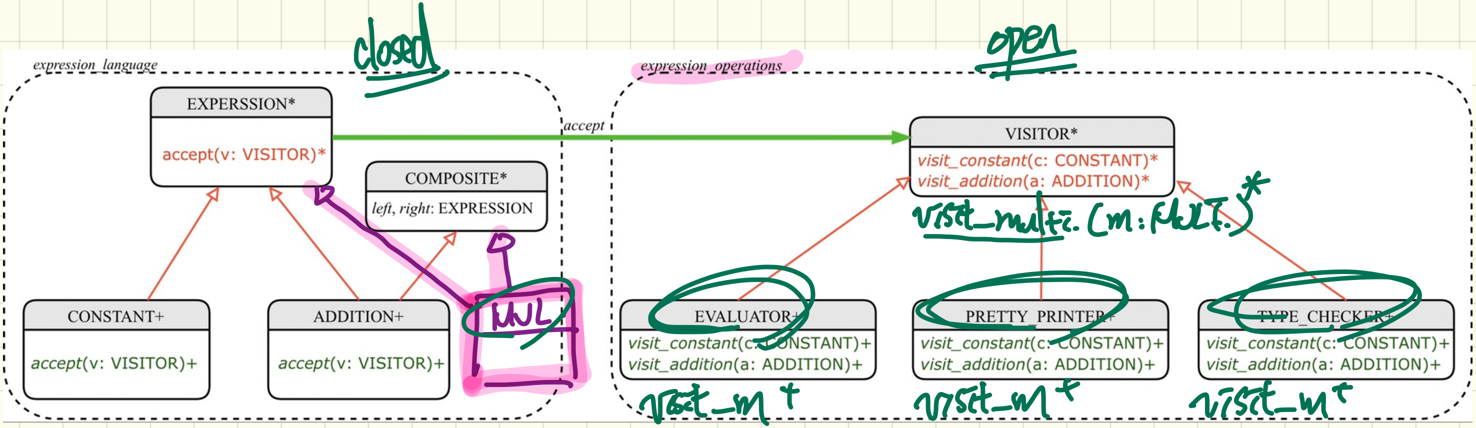
1st dispatch

Explain D.D. for ____ .

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

v: DT is EVALUATOR

Visitor Pattern: Open-Closed and Single-Choice Principles



MUL. Violates S.C.P.

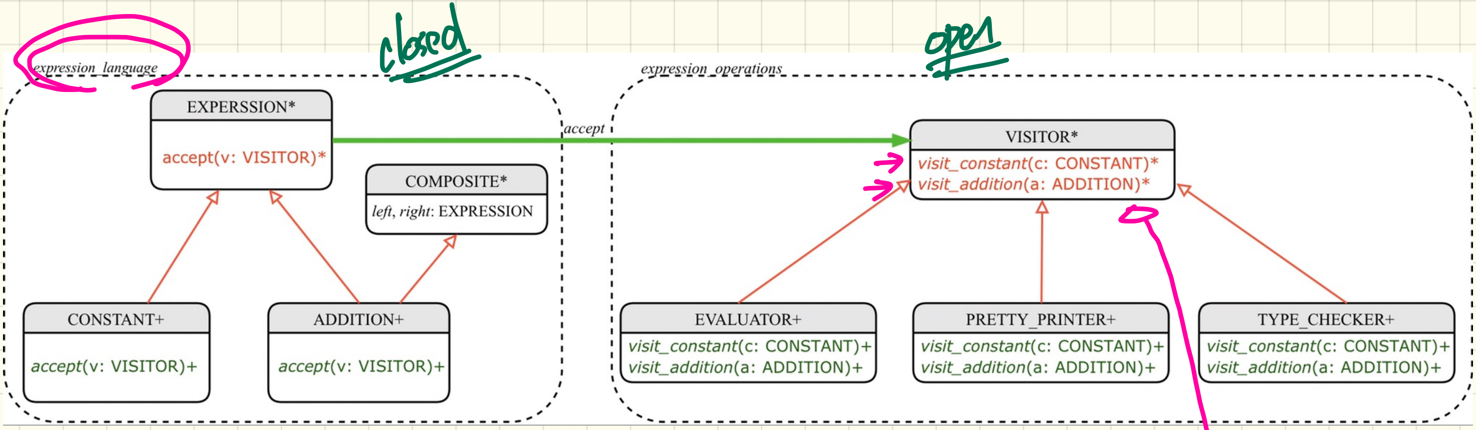
What if a new language construct is added?

↳ added to the closed part X

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

↳ language structure

Visitor Pattern: Open-Closed and Single-Choice Principles


closed
open
expression_language
EXPERSSION*
accept(v: VISITOR)*
COMPOSITE*
left, right: EXPRESSION
CONSTANT+
accept(v: VISITOR)+
ADDITION+
accept(v: VISITOR)+
accept
expression_operations
VISITOR*
visit_constant(c: CONSTANT)*
visit_addition(a: ADDITION)*
EVALUATOR+
visit_constant(c: CONSTANT)+
visit_addition(a: ADDITION)+
PRETTY_PRINTER+
visit_constant(c: CONSTANT)+
visit_addition(a: ADDITION)+
TYPE_CHECKER+
visit_constant(c: CONSTANT)+
visit_addition(a: ADDITION)+

CODE_GEN +
visit_c+
visit_a+

code-gen.

What if a new language operation is added?

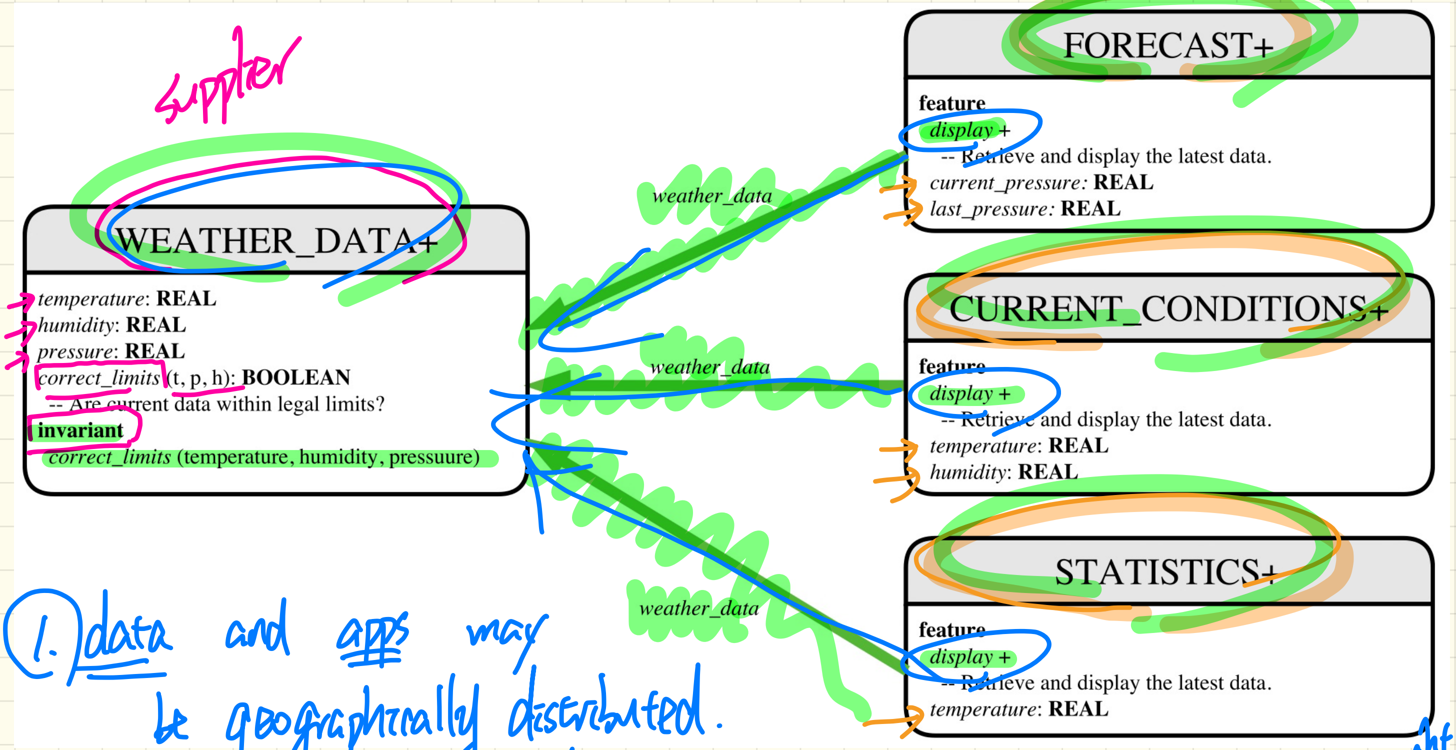
↳ satisfies S.C.P

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

↳ operation.

Weather Station: 1st Design

clients



supplier

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)

FORECAST+

feature

display +

-- Retrieve and display the latest data.

current_pressure: **REAL**

last_pressure: **REAL**

CURRENT_CONDITIONS+

feature

display +

-- Retrieve and display the latest data.

temperature: **REAL**

humidity: **REAL**

STATISTICS+

feature

display +

-- Retrieve and display the latest data.

temperature: **REAL**

① data and apps may be geographically distributed.

② when "display" is invoked, retrieve data (which might be [illegible])

- Observer
- Event-Driven Design

extra lecture

tentatively.

Friday

1pm - 2:30pm

- software verification

next M/W