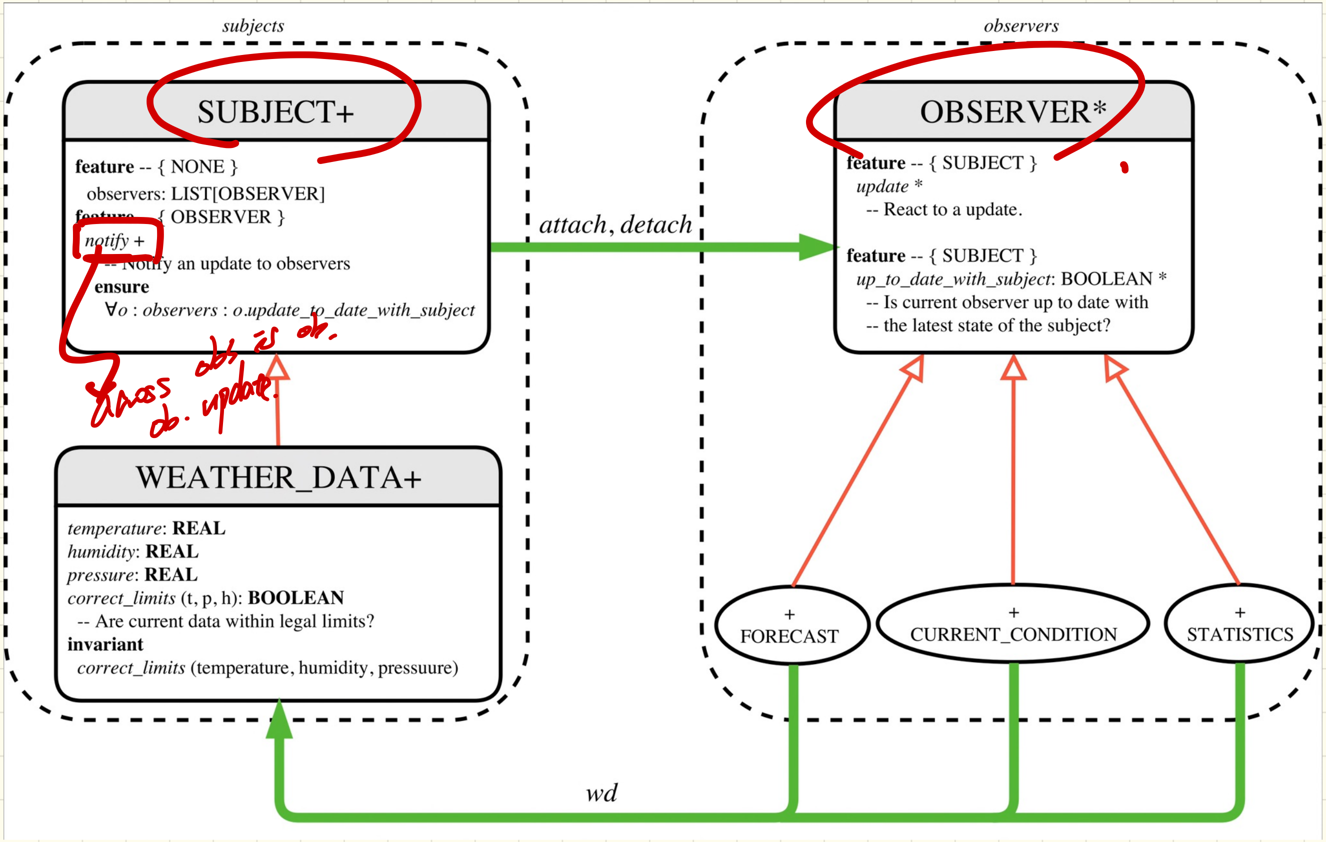
LECTURE 22

TUESDAY NOVEMBER 26

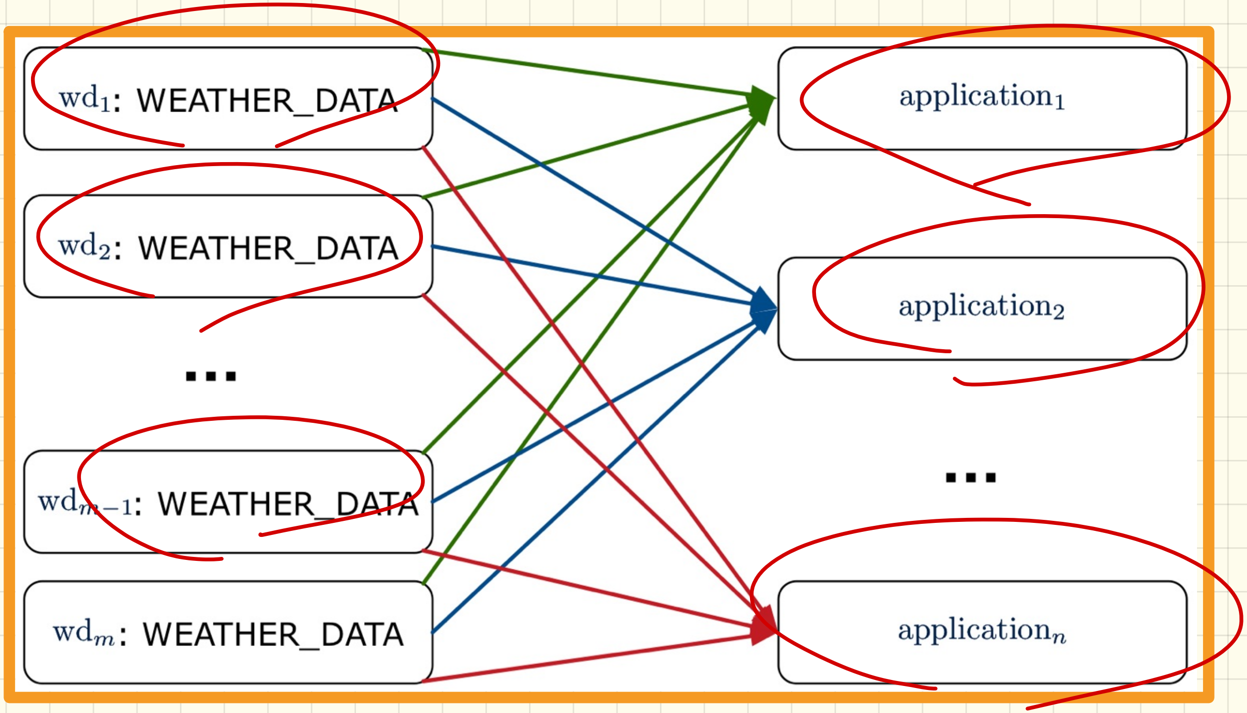
- REVIEW SESSIONS FOR EXAM
  SURVEY ON MOODLE

- MAKE-UP LECTURES:
  NOV. 15 } RECORDINGS
  NOV. 22 }

Observer Pattern: Application to Weather Station


subjects
observers
SUBJECT+
feature -- { NONE }
observers: LIST[OBSERVER]
feature -- { OBSERVER }
notify +
-- Notify an update to observers
ensure
∀o : observers : o.update_to_date_with_subject
across obs is ob.
ob. update.
attach, detach
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_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
+ CURRENT_CONDITION
+ STATISTICS
wd

Multiple Subjects vs. Multiple Observers: Observer Pattern

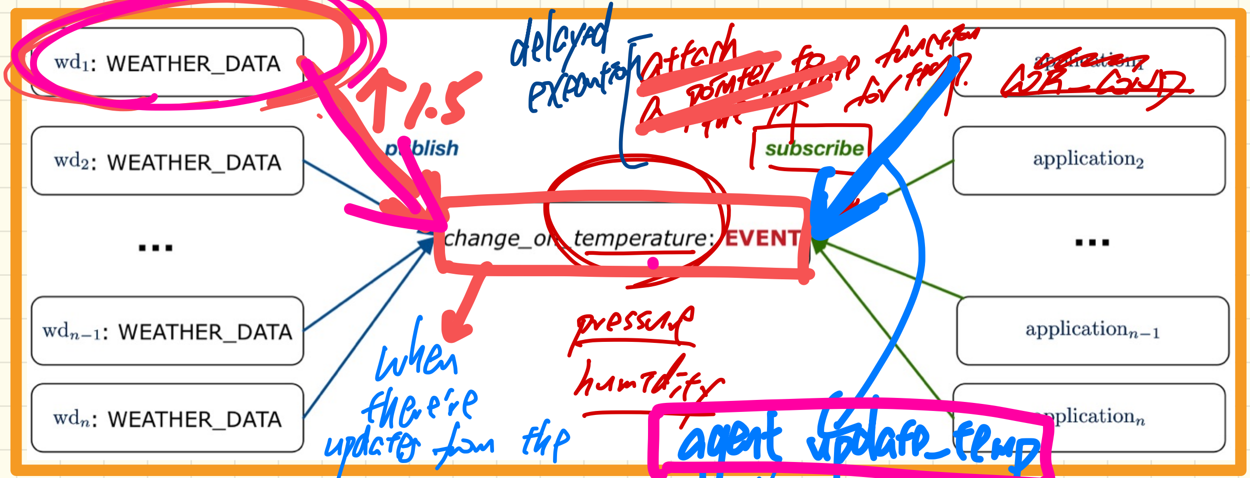


Q1. Overall **Complexity**?

Q2. **Complexity** of adding a new **subject**?

Q3. **Complexity** of adding a new **observer**?

Multiple Subjects vs. Multiple Observers: Event-Driven Design



when there're updates from the subject, the event will execute the pointer to functions stored previously.

Q1. Overall Complexity?

Q2. Complexity of adding a new subject?

Q3. Complexity of adding a new observer?

Q4. Complexity of adding a new event type?

① update_tempeatre

↳ 1. does not return anything

2. execution of u_t. right away.

② agent update_temperature

↳ PROCEDURE (for delayed execution).

Event-Driven Design in Eiffel

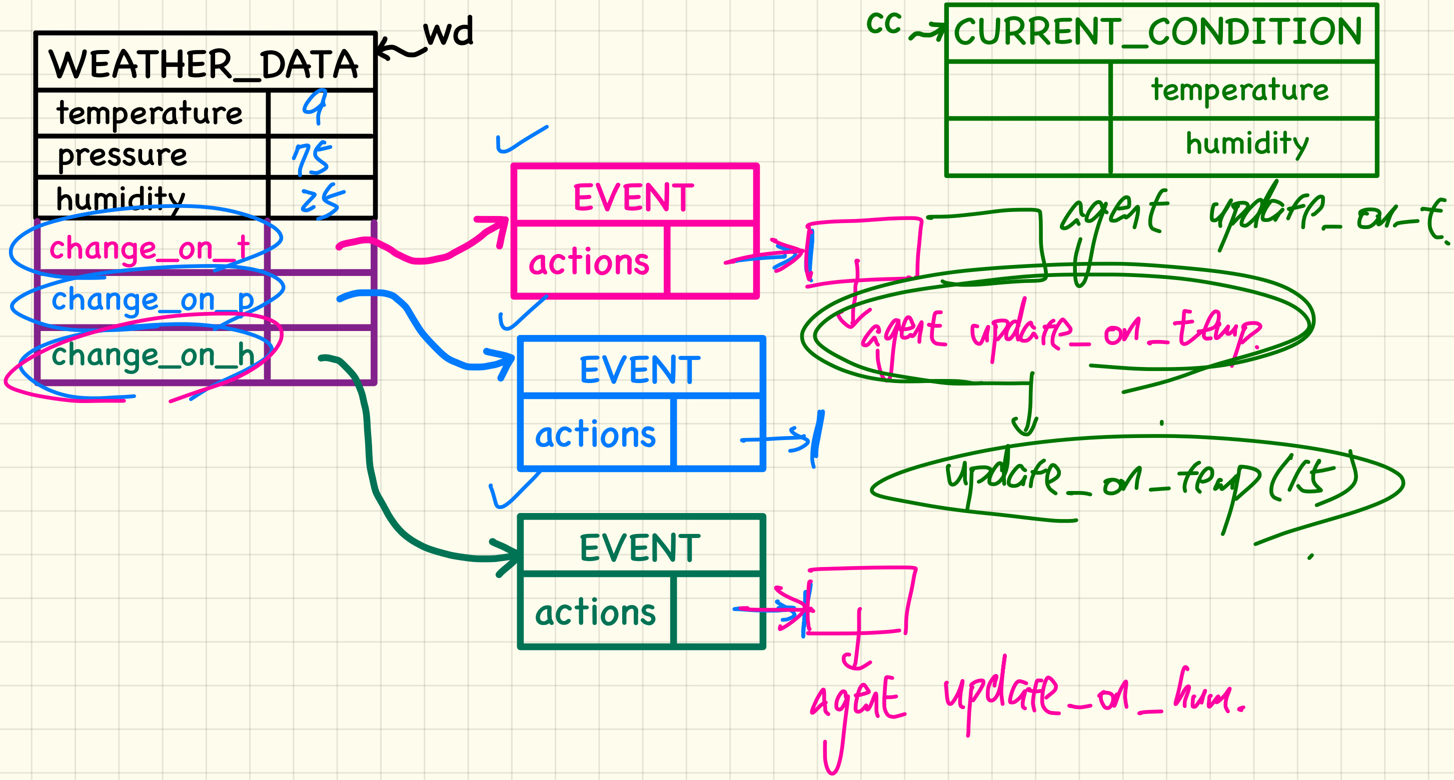
```
class WEATHER_STATION create make
feature
  cc: CURRENT_CONDITIONS
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd)
       wd.set_measurements (15, 60, 30.4)
       cc.display
       wd.set_measurements (11, 90, 20)
       cc.display
    end
end
```

```
class CURRENT_CONDITIONS
create make
feature -- Initialization
  make(wd: WEATHER_DATA)                                  EVENT
    do
      wd.change_on_temperature.subscribe (agent update_temperature)
      wd.change_on_temperature.subscribe (agent update_humidity)
    end                  humidity
feature
  temperature: REAL
  humidity: REAL
  update_temperature (t: REAL) do temperature := t end
  update_humidity (h: REAL) do humidity := h end
  display do ... end
end
```

```
class WEATHER_DATA
create make
feature -- Measurements
  temperature: REAL ; humidity: REAL ; pressure: REAL
  correct_limits(t,p,h: REAL): BOOLEAN do ... end
  make (t, p, h: REAL) do ... end
feature -- Event for data changes
  change_on_temperature : EVENT[TUPLE[REAL]]once create Result end
  change_on_humidity : EVENT[TUPLE[REAL]]once create Result end
  change_on_pressure : EVENT[TUPLE[REAL]]once create Result end
feature -- Command                  15  60  30.4
  set_measurements(t, p, h: REAL)
    require correct_limits(t,p,h)
    do temperature := t ; pressure := p ; humidity := h
      change_on_temperature.publish ([t])  (1) [15]
      change_on_humidity .publish ([p])
      change_on_pressure .publish ([h])
    end
invariant correct_limits(temperature, pressure, humidity) end
```

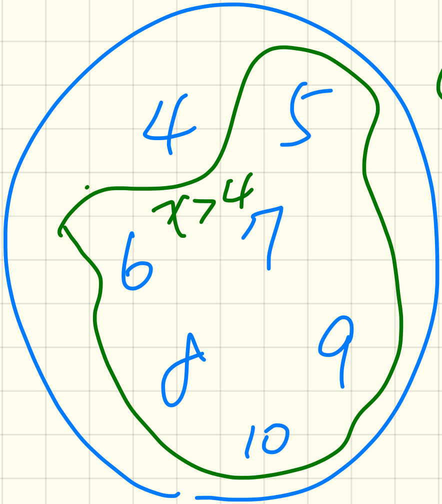
```
class EVENT [ARGUMENTS -> TUPLE ]
create make
feature -- Initialization          e1: EVENT[[REAL]]
                                   e2: EVENT[[S, I]]
  actions: LINKED_LIST[PROCEDURE[ARGUMENTS]]
  make do create actions.make end
feature
  subscribe (an_action: PROCEDURE[ARGUMENTS])
    require action_not_already_subscribed: not actions.h
    do actions.extend (an_action)
    ensure action_subscribed: action.has(an_action) end
  publish (args: G)  [15]
    do from actions.start until actions.after
      loop actions.item.call (args) ; actions.forth end
    end
                      A PROCEDURE  [15]
end
```

Event-Driven Design in Eiffel: Runtime



$\boxed{x > 3}$ allows more value (i.e. 4)

$x > 4$ stronger does not allow 4.



$\boxed{x > 4} \Rightarrow \boxed{x > 3}$

antecedent stronger

Consequence weaker.

Program Correctness: Example (1)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 3
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

$i = 4$

4

13 ~~i~~ := ~~i~~ + 9

13 ~~i~~ > 13

F

$\boxed{i > 3}$ → too weak

↳ it allows $i = 4$, which will cause postcondition violation.

Program Correctness: Example (2)

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 5
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

6
7
$\cancel{8}$

$i > 5 \Rightarrow i > 4$

Can any input value allowed by the precondition cause a postcondition?

No.

alternatively: $i > 4$

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 3
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

Hoare Triple

$\{ i > 3 \} \; i := i + 9 \; \{ i > 13 \}$

precondition

program/ implementation

postcondition

Boolean expression

Starting in a state that satisfies the precondition, if we execute imp., then it will ① terminate ② establish postcond.

```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > ~~3~~ ~~5~~ 4
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

$\{i > 4\}\ i := i + 9\ \{i > 13\}$ True

$\{i > 3\}\ i := i + 9\ \{i > 13\}$

↳ False ∵ $i = 4$ will be allowed by precondition

$\{i > 5\}\ i := i + 9\ \{i > 13\}$

↳ True ∵ all values allowed by $i > 5$ will establish $i > 13$ after executing $i := i + 9$.