# EECS3311 Software Design (Fall 2020)

## Q&A - Lecture Series W9

Monday, November 16

1. If we put

```
temperature:=weather_data.temperature
humidity:=weather_data.humidity
```

→ update.

in make of class CURRENT_CONDITIONS

Will both of them do the aliasing with weather_data's attributes?

Yes.

2. If wd.set_measurement(...) is called from clients,

Will it automatically update because of the aliasing?

↳ No. :( only weather data should change its measure. (cohesion)

3. If we just create a query to return weather_data.temperature and weather_data.humidity

And call the query in display

Isn't it save time and space for updating?

# Weather Station:

# 1st Implementation

```
class WEATHER_DATA create make
feature -- Data
  temperature: REAL
  humidity: REAL
  pressure: REAL
feature -- Queries
  correct_limits(t,p,h: REAL): BOOLEAN
    ensure
      Result implies -36 <=t and t <= 60
      Result implies 50 <= p and p <= 110
      Result implies 0.8 <= h and h <= 100
feature -- Commands
  make (t, p, h: REAL)
    require
      correct_limits(temperature, pressure, humidity)
    ensure
      temperature = t and pressure = p and humidity = h
invariant
  correct_limits(temperature, pressure, humidity)
end
```

```
class FORECAST create make
feature -- Attributes
  current_pressure: REAL
  last_pressure: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do last_pressure := current_pressure
       current_pressure := weather_data.pressure
    end
  display
    do update
```

```
class CURRENT_CONDITIONS create make
feature -- Attributes
  temperature: REAL
  humidity: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = wd
  update
    do temperature := weather_data.temperature
       humidity := weather_data.humidity
    end
  display
    do update
```

weather_data := wd

update necessary?

No, the 1st step of display invoke update anyway

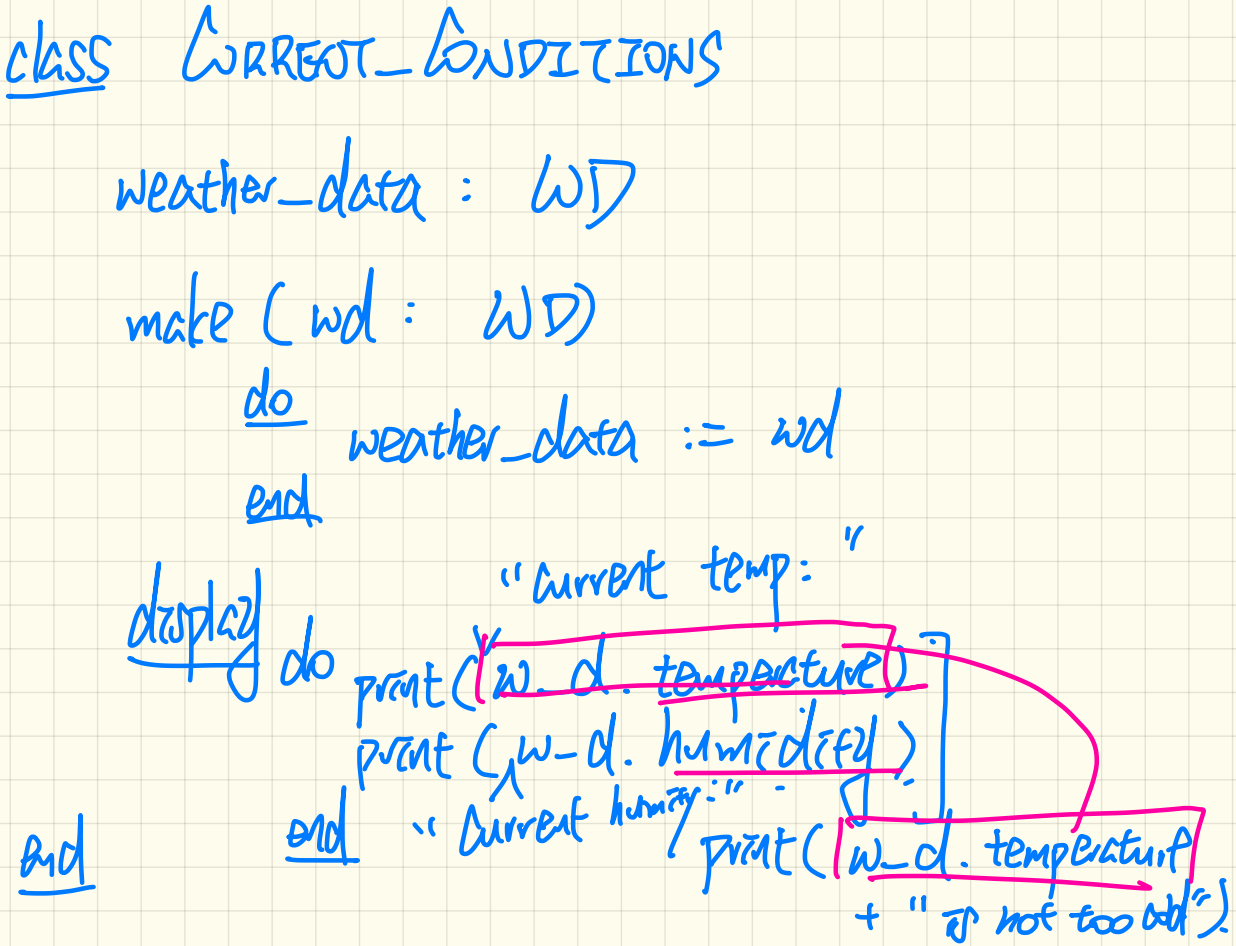
```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature
       -- Update min, max if necessary.
    end
  display
    do update
```

latest_temp.

each invocation to be a remote call.

may need

```
latest_temp: REAL
do
  Result := w_d.temp
end
```


class CURRENT_CONDITIONS
weather_data : WD
make (wd : WD)
do
weather_data := wd
end
display do
print ("current temp: ")
print (w_d. temperature)
print (w_d. humidity)
end "current humidity: "
print (w_d. temperature
+ "is not too odd")
end

# Weather Station:

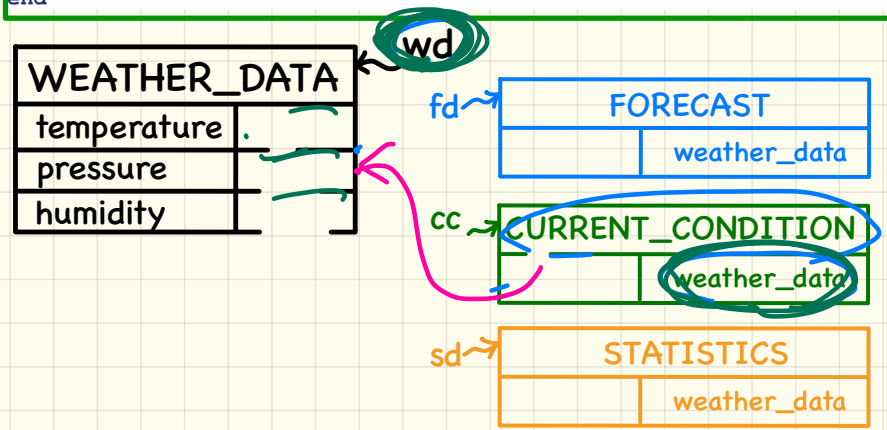
## Testing 1st Design

```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd) ; create fd.make (wd) ; create sd.make(wd)
       wd.set_measurements (15, 60, 30.4)
       cc.display ; fd.display ; sd.display
       cc.display ; fd.display ; sd.display
       wd.set_measurements (11, 90, 20)
       cc.display ; fd.display ; sd.display
  end
end
```

server.

cc.update

cc.weather_data := wd




```
class FORECAST create make
feature -- Attributes
  current_pressure: REAL
  last_pressure: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do last_pressure := current_pressure
       current_pressure := weather_data.pressure
    end
  display
    do update
```

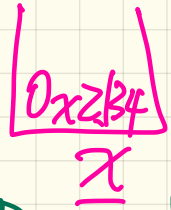
```
class CURRENT_CONDITIONS create make
feature -- Attributes
  temperature: REAL
  humidity: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = wd
  update
    do temperature := weather_data.temperature
       humidity := weather_data.humidity
    end
  display
    do update
```

```
class STATISTICS create make
feature -- Attributes
  weather_data: WEATHER_DATA
  current_temp: REAL
  max, min, sum_so_far: REAL
  num_readings: INTEGER
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = a_weather_data
  update
    do current_temp := weather_data.temperature
       -- Update min, max if necessary.
    end
  display
    do update
```

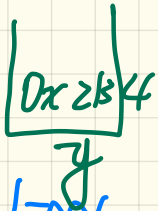
y : S_P
x : S_P

# aliasing.

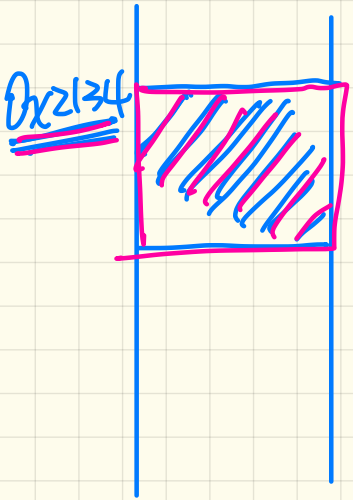
create x. make y := x

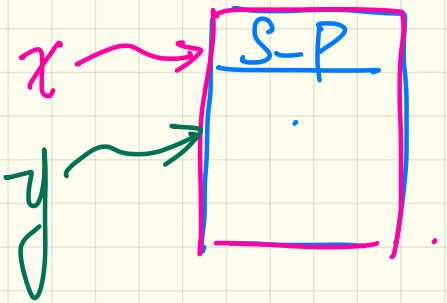



x=y T




multiple variables store the address of the same object



# Weather Station: Testing the Observer Pattern

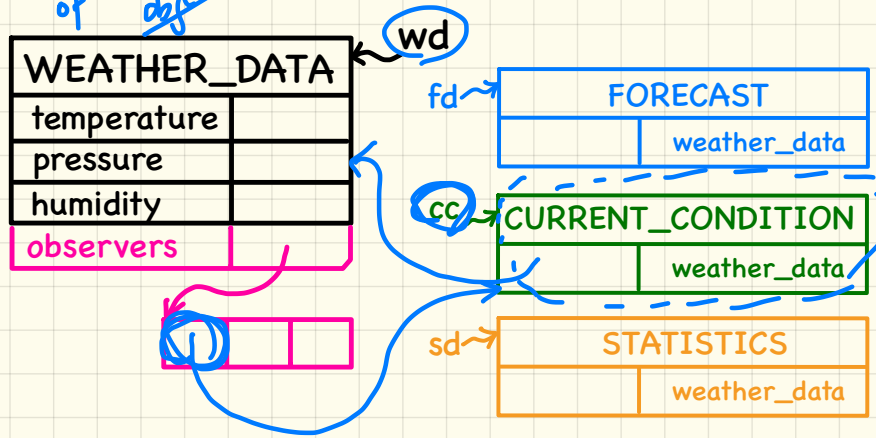
```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd) ; create fd.make (wd) ; create sd.make(wd)
       wd.set_measurements (15, 60, 30.4)
       wd.notify
       cc.display ; fd.display ; sd.display
       cc.display ; fd.display ; sd.display

       wd.set_measurements (11, 90, 20)
       wd.notify
       cc.display ; fd.display ; sd.display
    end
end
```

```
class FORECAST
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```

```
class CURRENT_CONDITIONS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```

```
class STATISTICS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```




why does passing "Current" inside the object's constructor make sense? "Current" object is not technically made yet as it is still executing its "make" constructor?

# Design 1 vs. Design 2: Up to Date?

```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd) ; create fd.make (wd) ; create sd.make(wd)

       wd.set_measurements (15, 60, 30.4)
       cc.display ; fd.display ; sd.display
       cc.display ; fd.display ; sd.display

       wd.set_measurements (11, 90, 20)
       cc.display ; fd.display ; sd.display
  end
end
```

①

```
class CURRENT_CONDITIONS create make
feature -- Attributes
  temperature: REAL
  humidity: REAL
  weather_data: WEATHER_DATA
feature -- Commands
  make(wd: WEATHER_DATA)
    ensure weather_data = wd
  update
    do temperature := weather_data.temperature
       humidity := weather_data.humidity
    end
  display
    do update
```

cc.display → up-to-date? YES.!

```
class WEATHER_STATION create make
feature -- Attributes
  cc: CURRENT_CONDITIONS ; fd: FORECAST ; sd: STATISTICS
  wd: WEATHER_DATA
feature -- Commands
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd) ; create fd.make (wd) ; create sd.make(wd)

       wd.set_measurements (15, 60, 30.4)
       wd.notify
       cc.display ; fd.display ; sd.display
       cc.display ; fd.display ; sd.display

       wd.set_measurements (11, 90, 20)
       wd.notify
       cc.display ; fd.display ; sd.display
  end
end
```

wd.notify → EDP

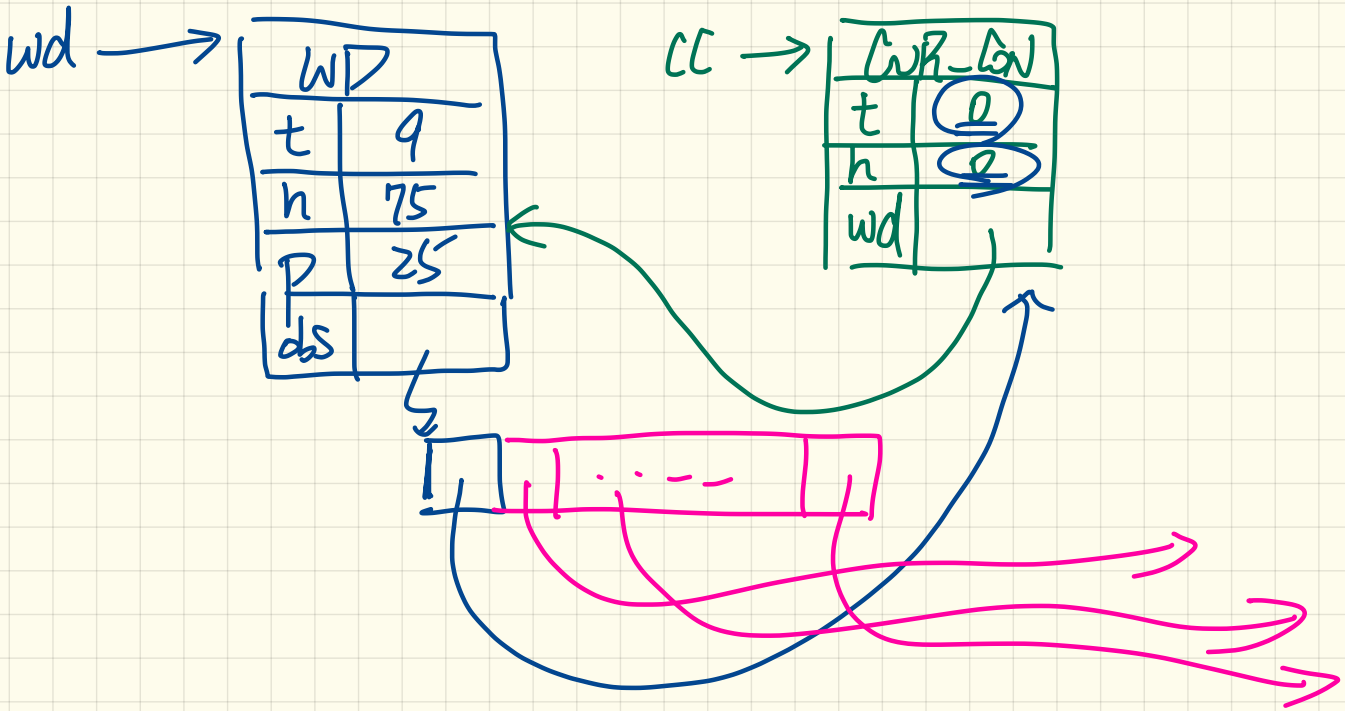
②

```
class CURRENT_CONDITIONS
inherit OBSERVER
feature -- Commands
  make(a_weather_data: WEATHER_DATA)
    do weather_data := a_weather_data
       weather_data.attach (Current)
    ensure weather_data = a_weather_data
           weather_data.observers.has (Current)
    end
```

wd → ① update.

cc.display → up-to-date? ② → wd.notify

Fix? Costly!! All obs. have to be updated. NO!


wd
WD
t
9
n
75
p
25
ds
CC
CWR_DW
t
n
wd

# Event-Driven Design: Multiple Subjects

Can you give an example of how the implementation of <Event> will be if there are 2 `subjects`?

wd1 publish → ch_on_temp ← wd2 publish
ch_on_temp → update CC, update SC1, update SC2

**Case 1**: Multiple instances of WEATHER_DATA

```
class EVENT [ARGUMENTS -> TUPLE]
create make
feature -- Initialization
  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)
    do from actions.start until actions.after
       loop actions.item.call (args) ; actions.forth end
    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
  set_measurements(t, p, h: REAL)
    require correct_limits(t,p,h)
    do temperature := t ; pressure := p ; humidity := h
      change_on_temperature.publish ([t])
      change_on_humidity.publish ([p])
      change_on_pressure.publish ([h])
    end
invariant correct_limits(temperature, pressure, humidity) end
```

Q. Which constructs ensures the same 3 Event instances are being published?
A. ONCE

wd1, wd2 : WEATHER_DATA

wd1 →

| W_D | |
|---|---|
| t | 10 |
| h | 20 |
| p | 30 |

wd2 →

| W_D | |
|---|---|
| t | 12 |
| h | 18 |
| p | 41 |

wd1.S_m(10, 20, 30)
wd2.S_m(12, 18, 41)

# Event-Driven Design: Multiple Subjects

Can you give an example of how the implementation of <Event> will be if there are 2 `subjects`?

**Case 2**: Multiple kinds of subjects: WEATHER_DATA, WEATHER_DATA_2

wd1 pub → C_O_T

```
class EVENT [ARGUMENTS -> TUPLE]
create make
feature -- Initialization
  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)
    do from actions.start until actions.after
       loop actions.item.call (args) ; actions.forth end
    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
  set_measurements(t, p, h: REAL)
    require correct_limits(t,p,h)
    do temperature := t ; pressure := p ; humidity := h
      change_on_temperature.publish ([t])
      change_on_humidity.publish ([p])
      change_on_pressure.publish ([h])
    end
invariant correct_limits(temperature, pressure, humidity) end
```

wd1 : WEATHER_DATA

wd2: WEATHER_DATA_2

wd1. set_m (10, 20, 30)

wd2. set_m (12, 18, 41)

```
class WEATHER_DATA_2
  set_m ( ... )
  do
  end
end
```

[? access to events?

# Singleton Pattern

(adapted).

expa:

class Utility

```
c_on_tem : EVENT[TUPLE]
    once
        create Result.make
    end

c_on_press: EVENT[TUPLE]
    once
        create Result.make
    end

end
```

ea: UTILITY

ea.c_on_temp.publish(...)

Both solutions satisfy cohesion.

# STATIC ROUTINES

```
class UTILITY

    c_on_tem : EVENT[TUPLE]
        once
            create Result.make
        ensure
            class
        end

end
```

Use.

{UTILITY}.c_on_item.publish(...)

Q1. If I understood it correctly,
Methodhandler in Java or Procedure in Eiffel is a library class which helps us to store functions/features into an object so that they are easier to be stored and retrieved?

↗ pointers *

↓ for delayed execution.

Q9.

Q2. In terms of our quiz this week, do we need to study event-driven design in only Eiffel syntax or do we also have to study event-driven design in Java syntax too?

→ once agent.

↓
1. do not memorize any code.
2. understand how classes work together
3. if any questions on Java, API will be given to you.

random number generator.

↓

deterministically.