EECS3311 Software Design (Fall 2020)

Q&A - Lecture Series W9

Tuesday, November 17

Weather Station:

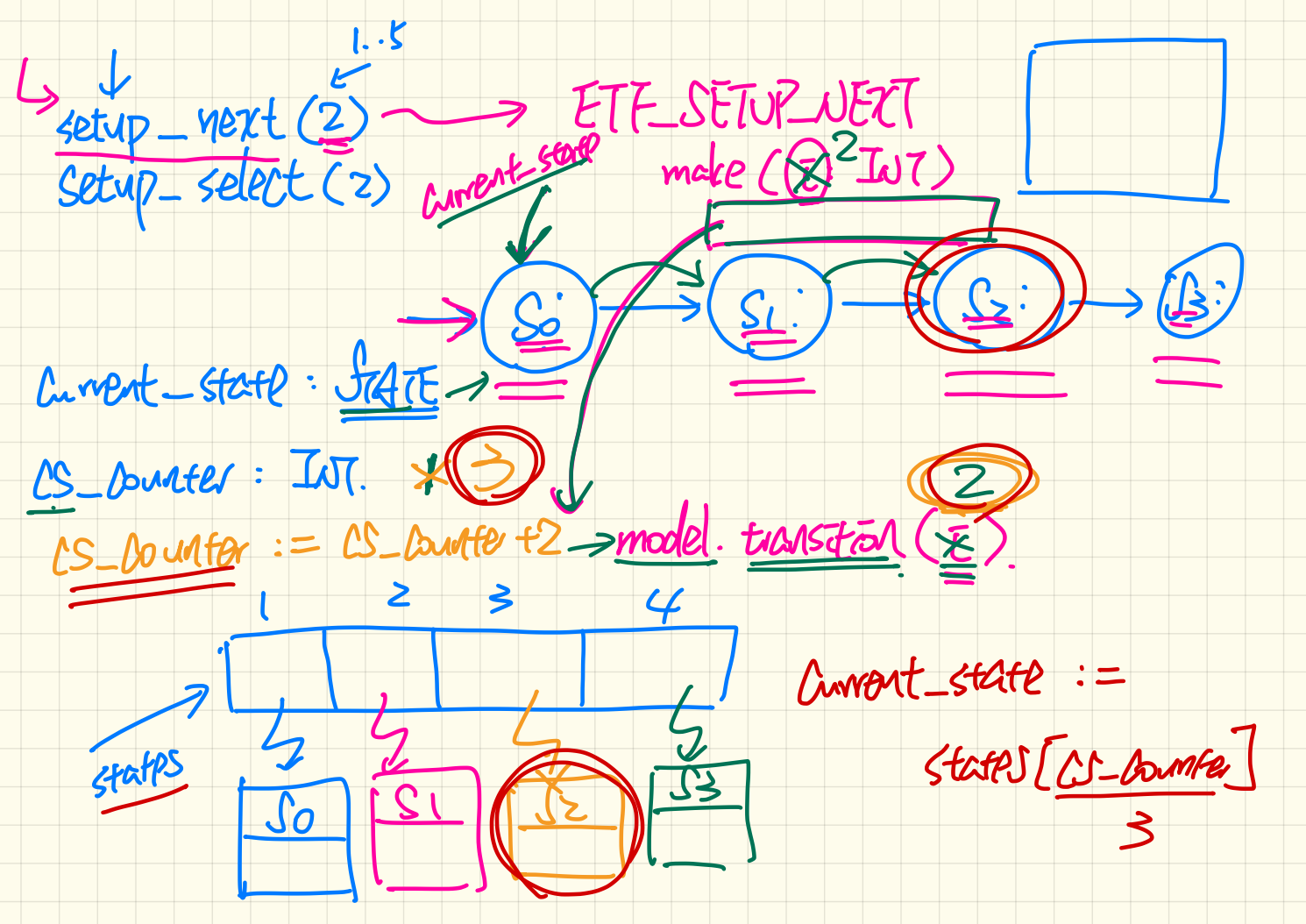
1st Implementation

```
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
```

```
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
```


setup_next (2)
1..5
setup_select (2)
ETF_SETUP_NEXT
make (?2 INT)
current_state
S0
S1
S2
S3
current_state : STATE
CS_counter : INT.
3
CS_counter := CS_counter +2
model. transition (?)
2
1
2
3
4
states
S0
S1
S2
S3
current_state :=
states [CS_counter]
3

Event-Driven Design

```
class EVENT [ ARGUMENTS -> TUPLE ]
    actions : LIST [ PROCEDURE [ ARGUMENTS ] ]
end
```

name of type parameter

constrained genericity.

use of the parameter

class EVENT < H extends TUPLE > Use.

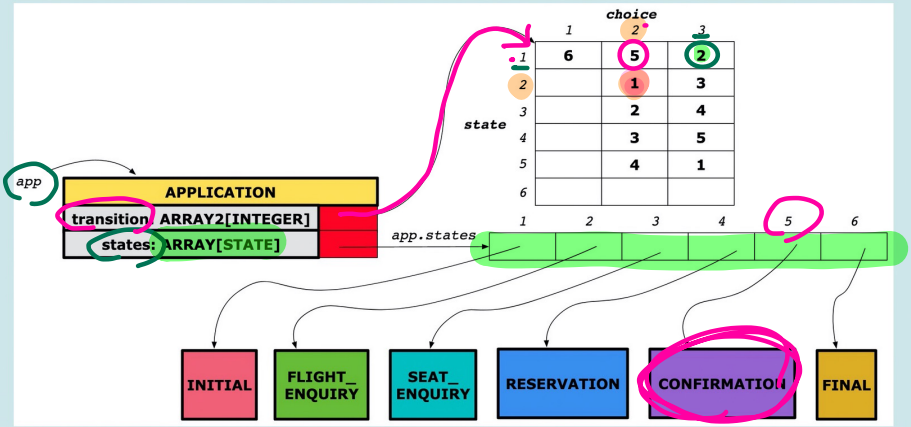
ch_on_temp : EVENT [INT]

INT not a descendant of TUPLE ✗

ch_on_temp : EVENT [TUPLE [REAL]]

descendant of TUPLE.

As discussed in the lecture, assume the following runtime setup of a client using the state design pattern:



That is, the above diagram suggests that a variable `**app: APPLICATION**` has been properly initialized.

Assume the following variable declaration:

current_state: STATE

Now consider the following re-assignment to `current_state` through a number of state transitions (according to the above runtime setup):

current_state := app.states [app.transition.item(app.transition.item(app.transition.item(1, 3), 2), 2)]

current_state := app.states [5]

DT: CONFIRMATION

attached {CONT} [-S]

attached {STATE} [-S]

Consider the template design pattern discussed in the lecture. For each of the following descriptions, determine whether it is **true** or **false**.

1. In the STATE class, the structure of a pattern is implemented as an effective routine, referencing all other deferred routines corresponding to components of the pattern.
2. In each of the effective child classes of STATE, it is expected that the inherited `execute` pattern is redefined to suit that specific state's need.
3. In the STATE class, components of a pattern are implemented as effective routines, expected to be reused in all effective descendant classes.
4. In the STATE class, components of a pattern are declared as deferred routines, expected to be implemented in all effective descendant classes.
5. In the STATE class, the structure of a pattern is declared as a deferred routine, expected to be implemented in all effective descendant classes.
6. In each of the effective child classes of STATE, it is expected that the inherited routines (e.g., `display`), corresponding to components of the pattern, are not redefined in order to maximize reusability.