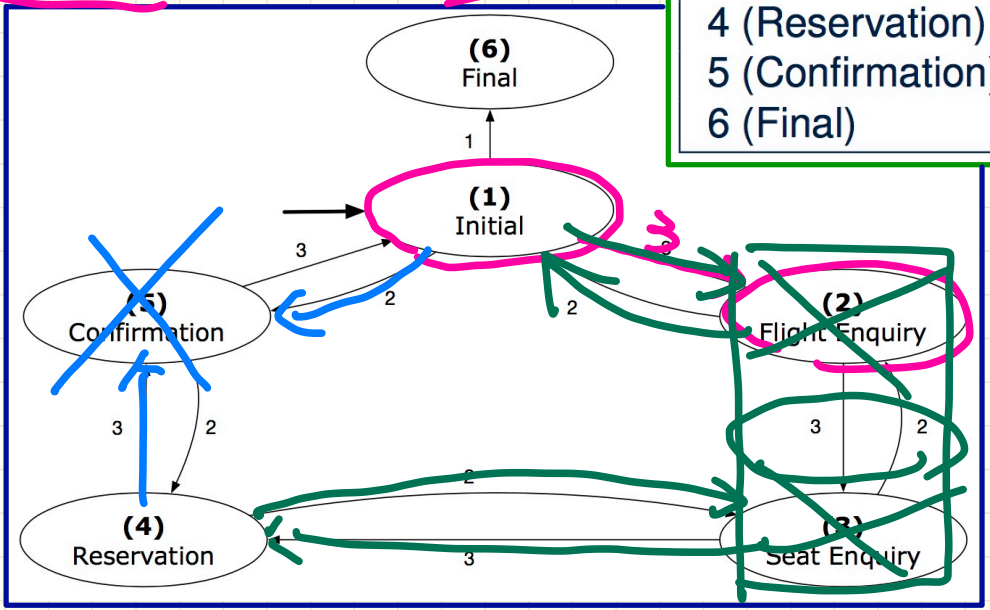
Lecture 18
Wednesday March 11

- Lab4 extended until 11am on Monday
- TA Hours: 9:30 to 11:30 on Thursday
- Office hours today shifted: 1pm to 3pm on Friday
    * Lab4
    * Labtest2
    * Exam

Finite State Machine (FSM)

model

State Transition Diagram



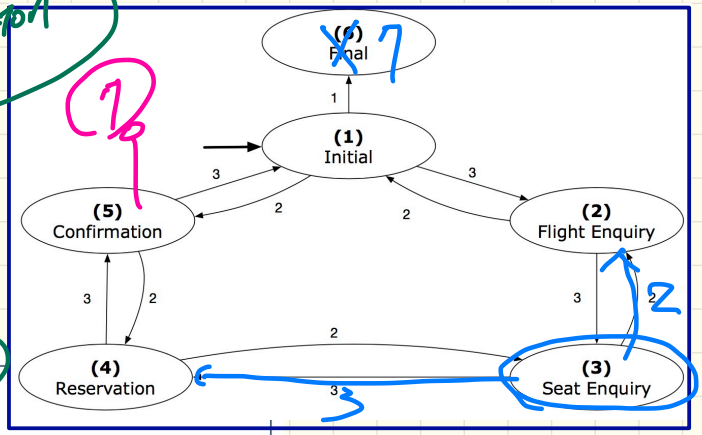
ARRAY2.

State Transition Table

SRC STATE \ CHOICE	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

ARRAY[ARRAY[...]]

Design of a Reservation System: First Attempt




```
1_Initial_panel:
  -- Actions for Label 1.
2_Flight_Enquiry_panel:
  -- Actions for Label 2.
3_Seat_Enquiry_panel:
  -- Actions for Label 3.
4_Reservation_panel:
  -- Actions for Label 4.
5_Confirmation_panel:
  -- Actions for Label 5.
6_Final_panel:
  -- Actions for Label 6.
```

```
3_Seat_Enquiry_panel:
from
   Display Seat Enquiry Panel
until
   not (wrong answer or wrong choice)
do
   Read user's answer for current panel
   Read user's choice C for next step
   if wrong answer or wrong choice then
      Output error messages
   end
end
Process user's answer
case C in
   2: goto 2_Flight_Enquiry_panel
   3: goto 4_Reservation_panel
end
```

Single Choice Principle

Design of a Reservation System: Second Attempt (1)

```
transition (src: INTEGER; choice: INTEGER): INTEGER
    -- Return state by taking transition 'choice' from 'src' state.
  require valid_source_state: 1 ≤ src ≤ 6
          valid_choice: 1 ≤ choice ≤ 3
  ensure valid_target_state: 1 ≤ Result ≤ 6
```

transition(3, 3) ⇝ 4

Examples:

transition(3, 2)

transition(3, 3)

transition : ARRAY2 [INT]

transition (3, 3) ↳ 4

State Transition Table

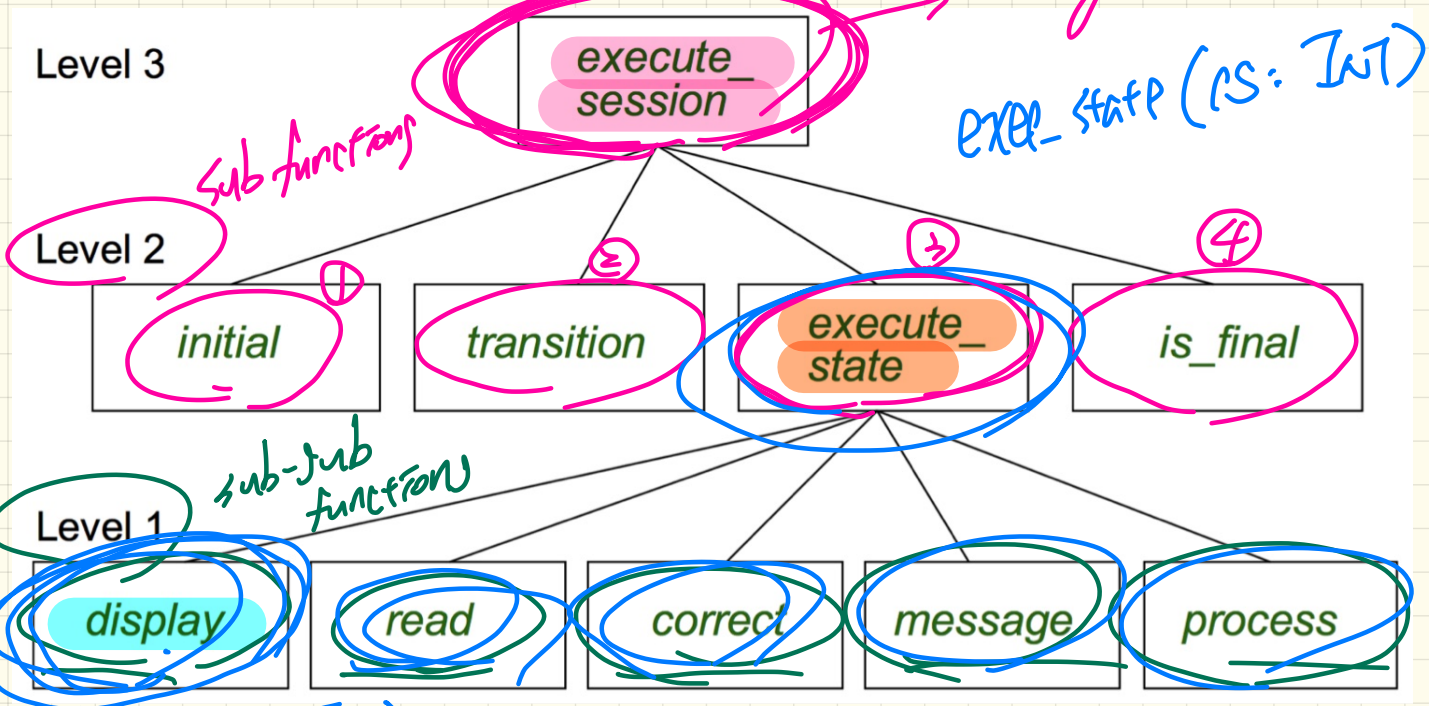
SRC STATE \ CHOICE	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

2D Array Implementation

state \ choice	1	2	3
1	6	5	2
2	X	1	3
3	X	2	4
4	X	3	5
5	X	4	1
6	X	X	

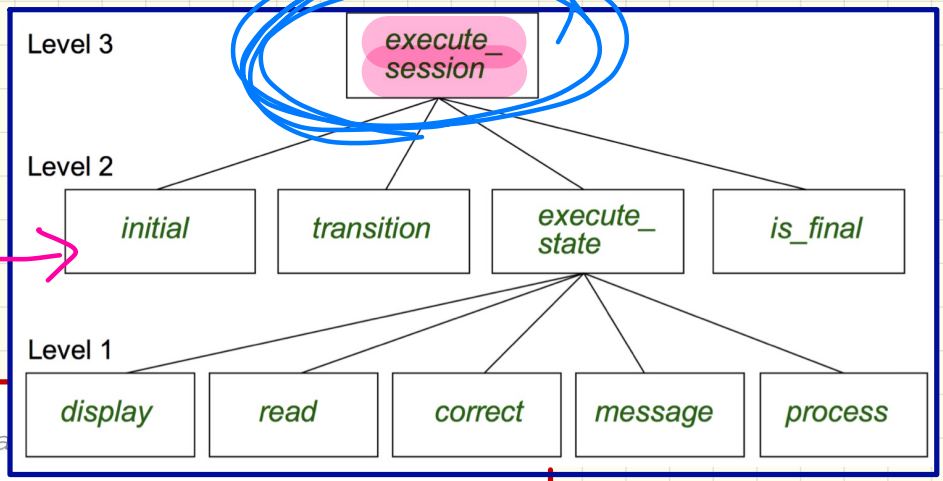
Design of a Reservation System: Second Attempt (2)

A Top-Down & Hierarchical Design



Design of a Reservation System: Second Attempt (3)

act (e: EXCEPTION)




```
execute_session
    -- Execute a full intera
  local
    current_state, choice: INTEGER
  do
    from
      current_state := initial
    until
      is_final (current_state)
    do
      choice := execute_state (current_state)
      current_state := transition (current_state, choice)
    end
  end
```

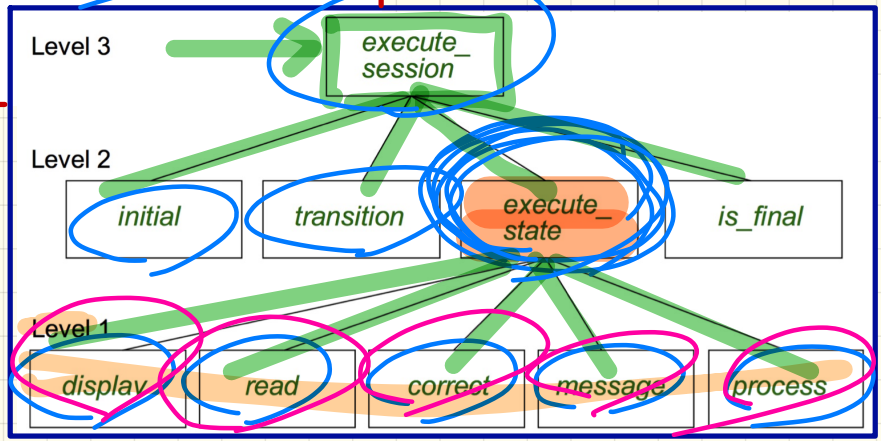
state-specific actions

Design of a Reservation System: Second Attempt (4)

```
execute_state (current_state: INTEGER): INTEGER
  -- Handle interaction at the current state.
  -- Return user's exit choice.
local
  answer: ANSWER; valid_answer: BOOLEAN; choice: INTEGER
do
  from
  until
    valid_answer
  do
    display(current_state)
    answer := read_answer(current_state)
    choice := read_choice(current_state)
    valid_answer := correct(current_state, answer)
    if not valid_answer then message(current_state, answer)
  end
  process(current_state, answer)
  Result := choice
end
```

pattern of interaction in each state (template)

e.s
t
m.
e.s.



add state 7.

display (cs : INT)

do

if cs = 1 then

elseif cs = 2 then

elseif cs = 6 then

elseif cs = 7 then

end

delete state 2.

read (cs : INT)

do

if cs = 1 then

elseif cs = 2 then

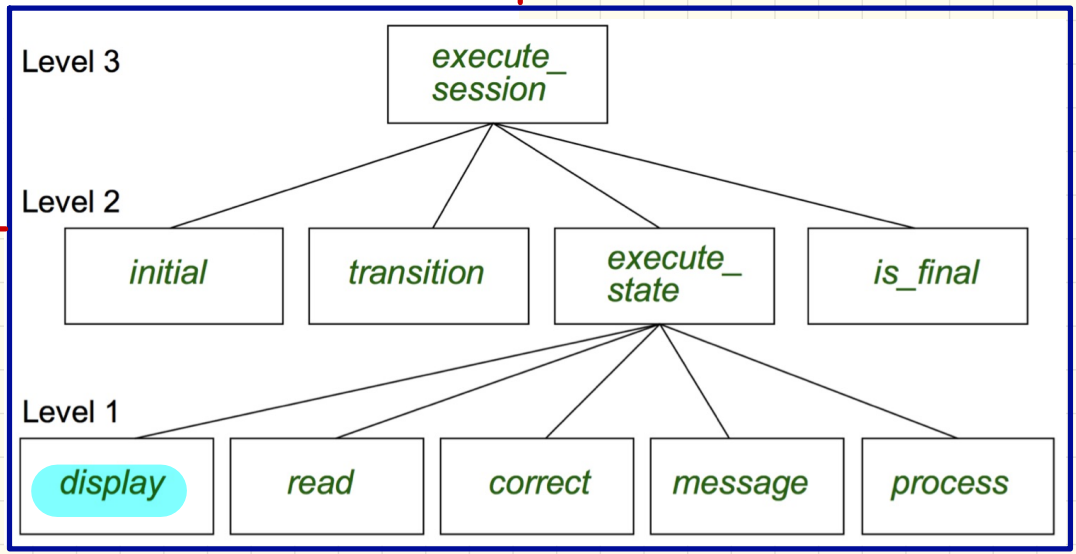
elseif cs = 6 then

elseif cs = 7 then

end

Design of a Reservation System: Second Attempt (5)

```
display(current_state: INTEGER)
  require
    valid_state: 1 ≤ current_state ≤ 6
  do
    if current_state = 1 then
      -- Display Initial Panel
    elseif current_state = 2 then
      -- Display Flight Enquiry Panel
    ...
    else
      -- Display
    end
  end
```



2nd Design Attempt

```
class
   STUDENT
create
   make
feature -- attribures
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
   premiumRate: REAL
   discountRate: REAL
feature -- command
   make (kind: INTEGER)
     do
       kind := a_kind
     end
   ...
end
```

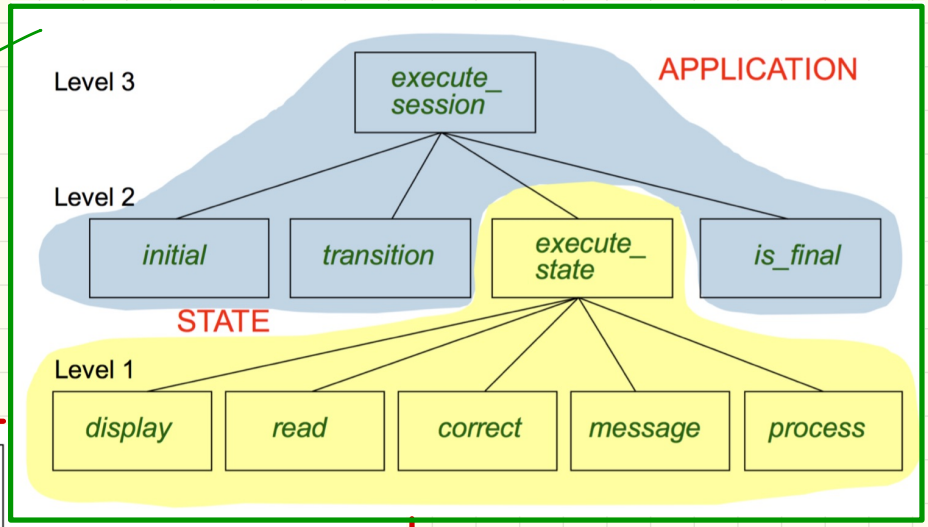
```
get_tuition: REAL
   local
      tuition: REAL
   do
      across courses is c loop
         tuition := tuition + c.fee
      end
      if kind = 1 then
         Result := tuition * premiumRate
      elseif kind = 2 then
         Result := tuition * discountRate
      end
   end
```

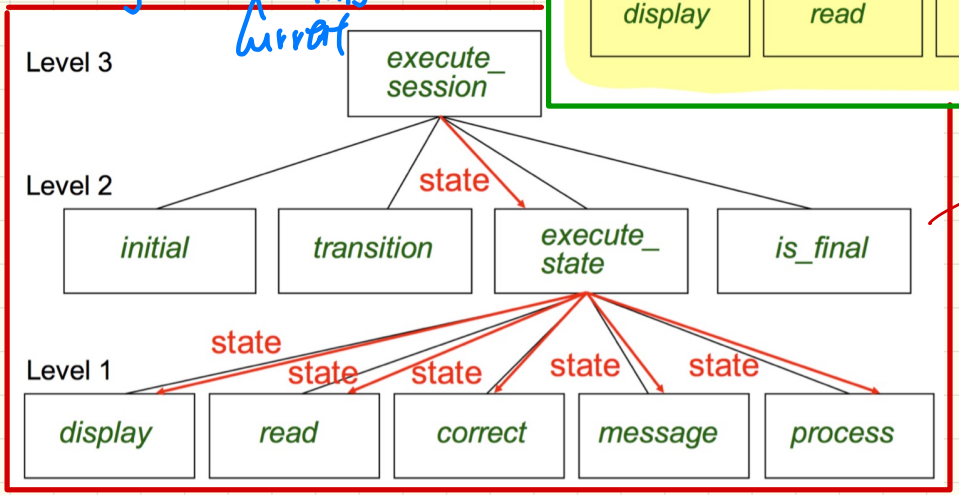
```
register (c: COURSE)
   local
      max: INTEGER
   do
      if kind = 1 then MAX := 6
      elseif kind = 2 then MAX := 4
      end
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

Moving from Top-Down Design to OO Design

Object-Oriented

current_state: STATE
current_state.*execute_session*

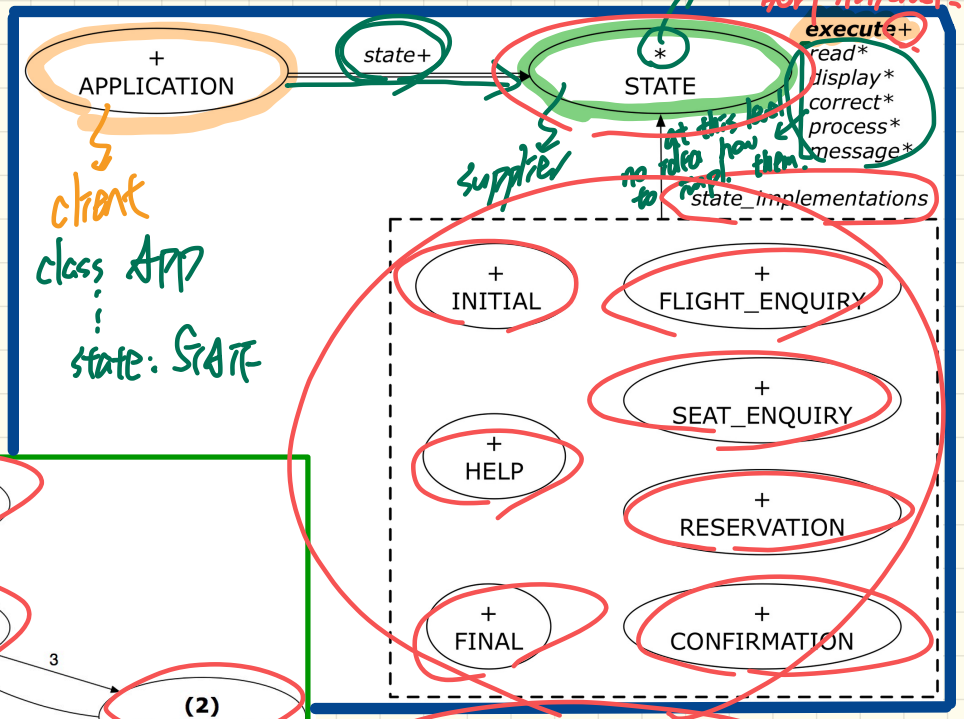


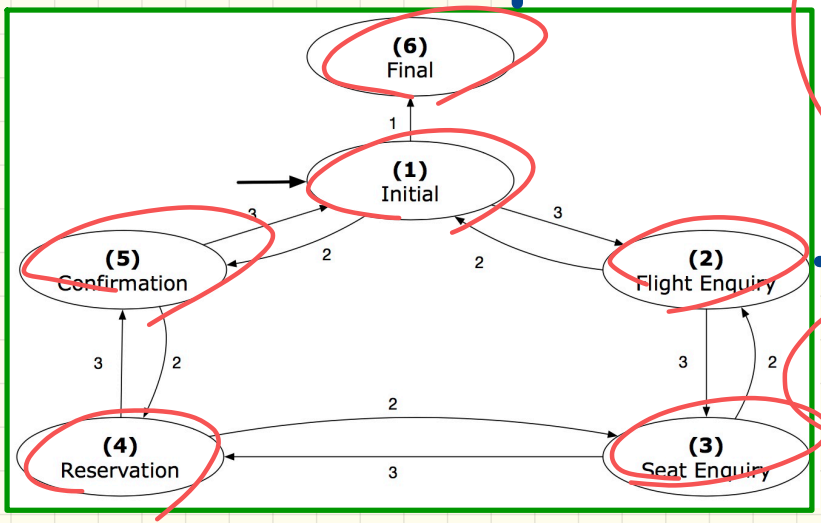


Top-Down

current_state: INTEGER
execute_session(current_stste)

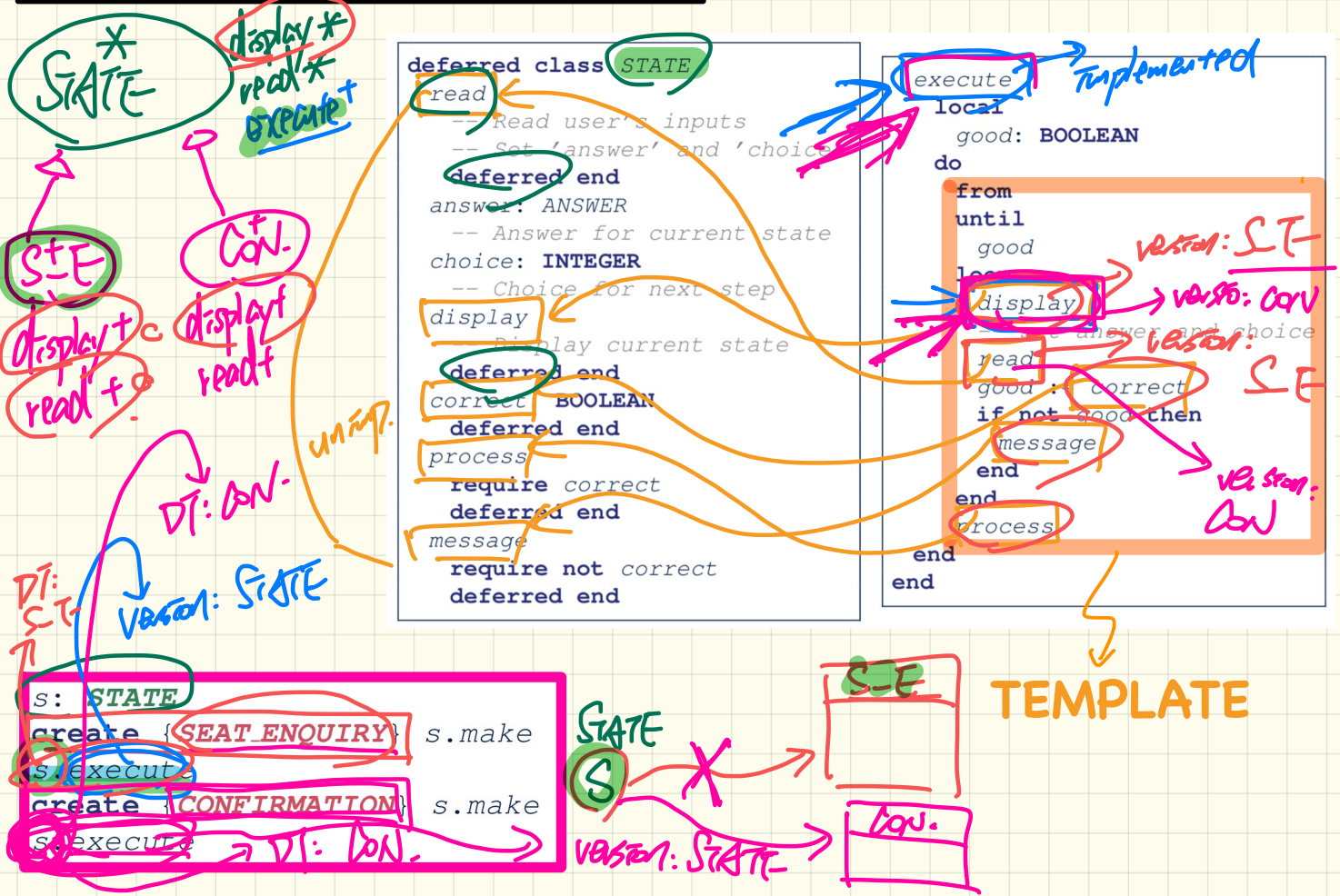
State Pattern: Architecture





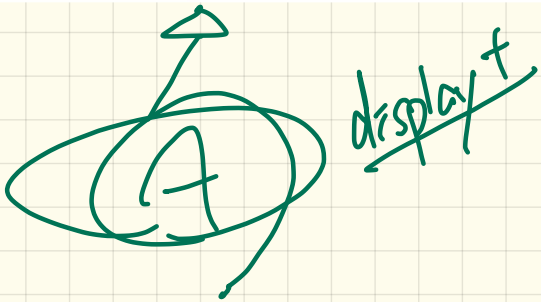

```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

State Pattern: State Module



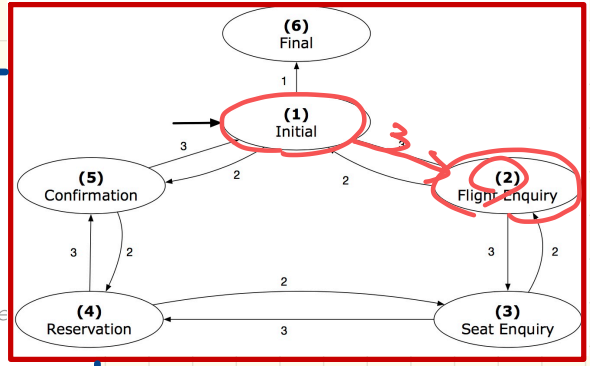
```
deferred class STATE
read
  -- Read user's inputs
  -- Set 'answer' and 'choice'
  deferred end
answer: ANSWER
  -- Answer for current state
choice: INTEGER
  -- Choice for next step
display
  -- Display current state
  deferred end
correct: BOOLEAN
  deferred end
process
  require correct
  deferred end
message
  require not correct
  deferred end
```

```
execute
  local
    good: BOOLEAN
  do
    from
    until
      good
    loop
      display
      -- Set answer and choice
      read
      good := correct
      if not good then
        message
      end
    end
    process
  end
end
```



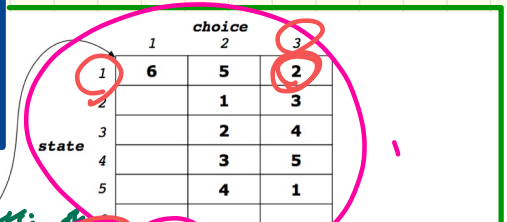
State Pattern: Test

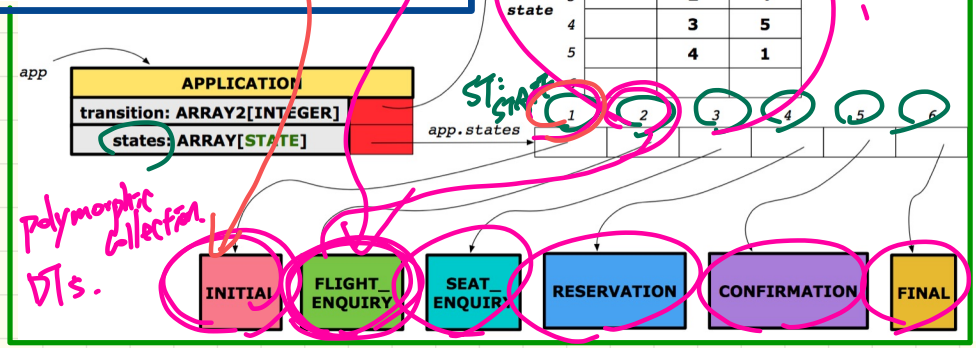
```
test_application: BOOLEAN
  local
    app: APPLICATION ; current_state: STATE ; index: INTEGER
  do
    create app.make (6, 3)
    app.put_state (create {INITIAL}.make, 1)
    -- Similarly for other 5 states.
    app.choose_initial (1)
    -- Transit to FINAL given current state INITIAL and choice
    app.put_transition (6, 1, 1)
    -- Similarly for other 10 transitions.

    index := app.initial
    current_state := app.states [index]
    Result := attached {INITIAL} current_state
    check Result end
    -- Say user's choice is 3: transit from INITIAL to FLIGHT_STATUS
    index := app.transition.item (index, 3)
    current_state := app.states [index]
    Result := attached {FLIGHT_ENQUIRY} current_state
end
```



transition (1, 3) → 2

states : ARRAY[STATE]





CS : SEAT_

DT : RES.

DT : F_E

CS := states [2]

CS. display (1) → F_E

CS := states [4]

CS. display (2) → RES