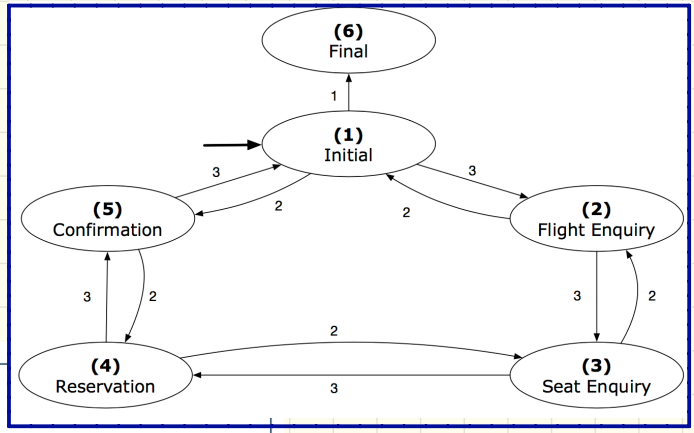
Monday March 18

Lecture 18

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.
```

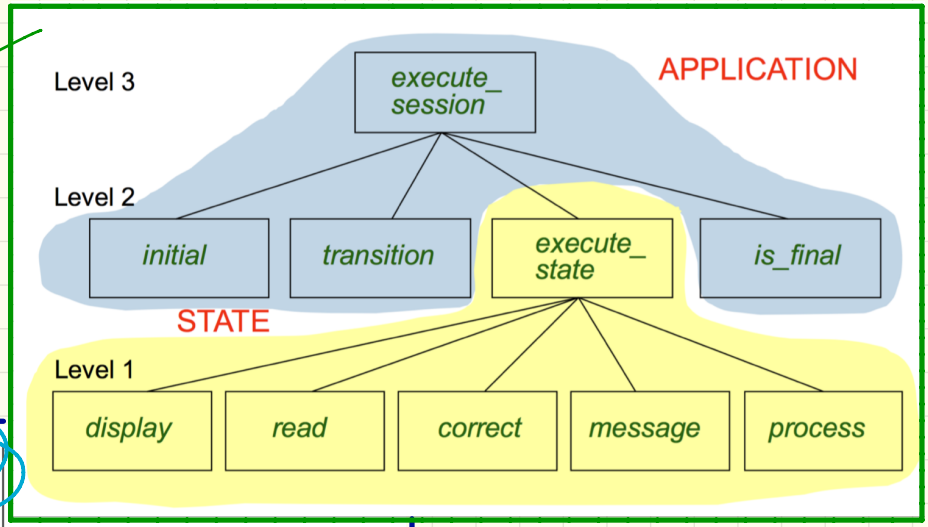
```
3_Seat_Enquiry_panel:
from
  Display Seat Enquiry Panel
until
  not (wrong answer or wrong choice)
do
  Read user's answer for current panel
  Read user's choice C for next step
  if wrong answer or wrong choice then
    Output error messages
  end
end
Process user's answer
case C in
  2: goto 2_Flight_Enquiry_panel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

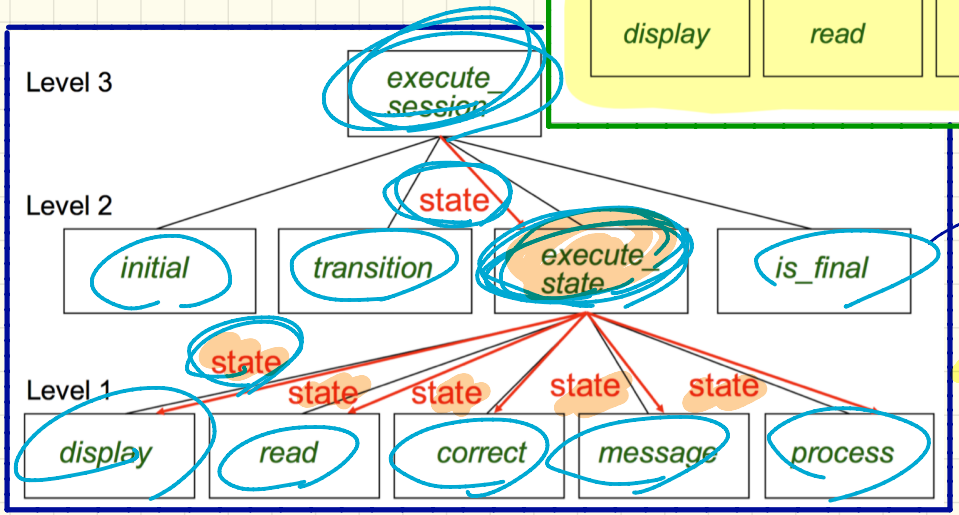
Moving from Hierarchical Design to OO Design

OO

current_state : STATE

current_state . execute_session





HIERARCHICAL

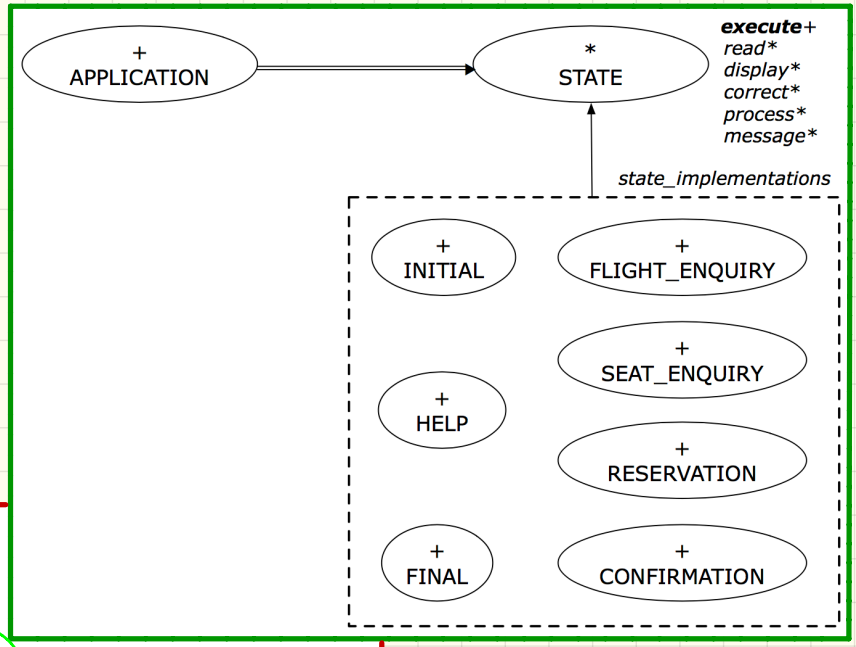
current_state : INTEGER

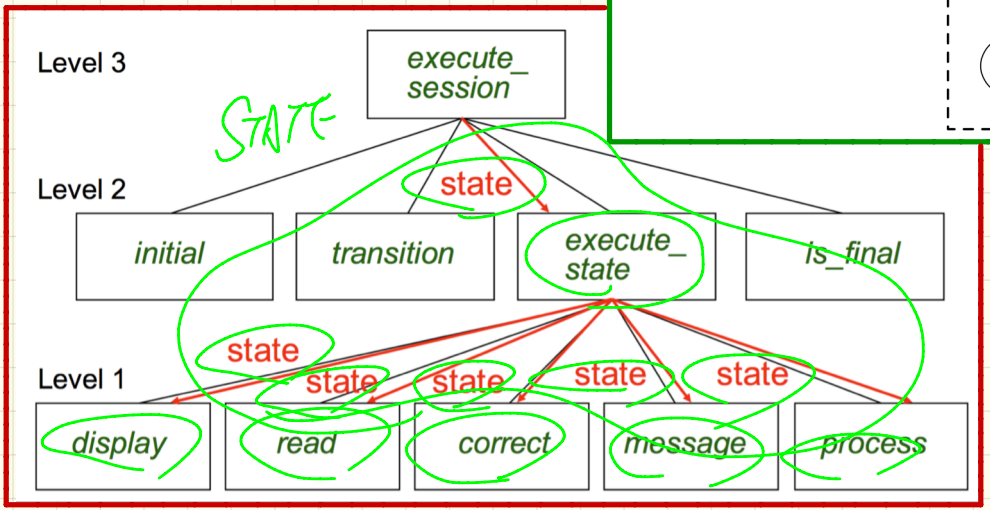
execute_session (current_state)

Interactive System: Non-OO vs. OO

current_state : STATE

current_state.execute_session





current_state : INTEGER

execute_session (current_state)

Non-OO

→ execute_session (cs: INT)

do

display (cs)

read_answer (cs)

end

→ s1. execute

→ s2. execute

OO (State Pattern)

class STATE

execute

do

s1 current.display

s2

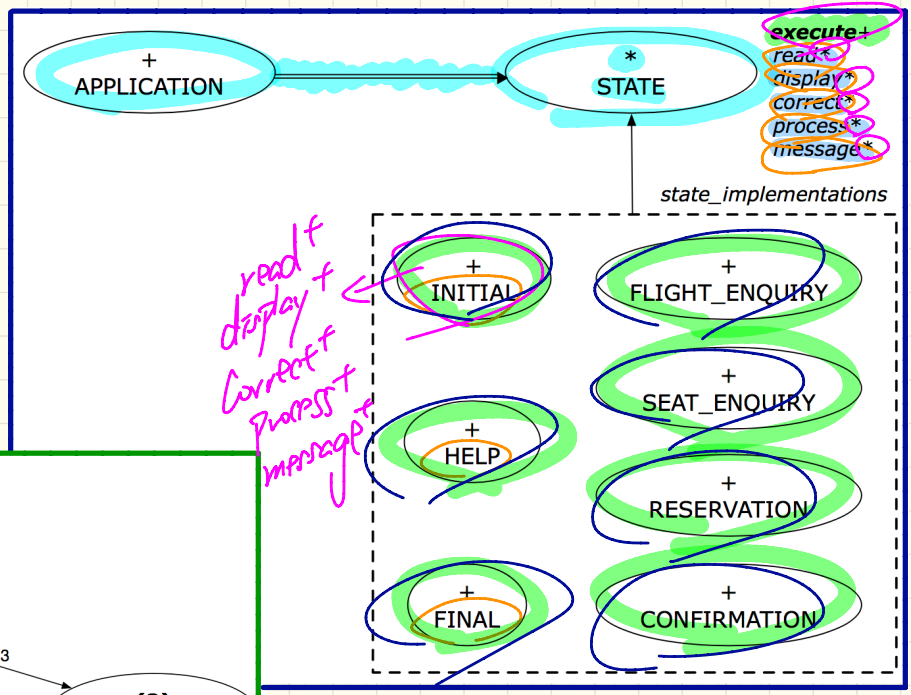
s1 current.read_answer

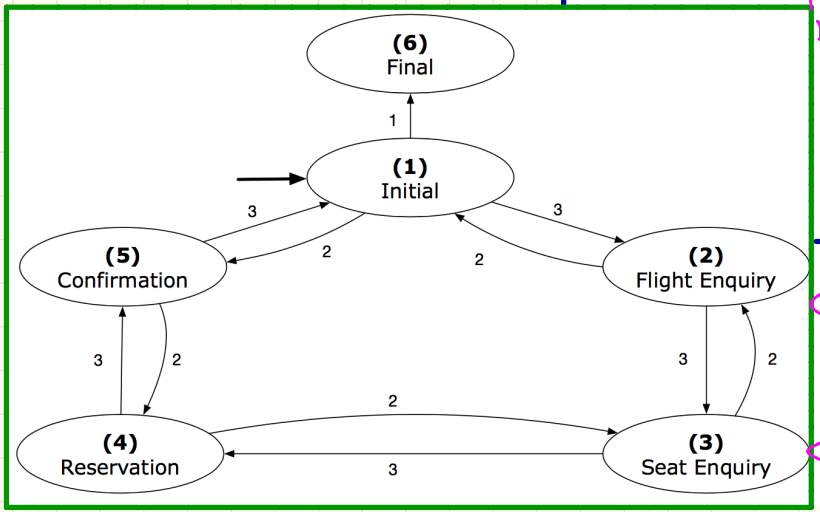
s2

end

end

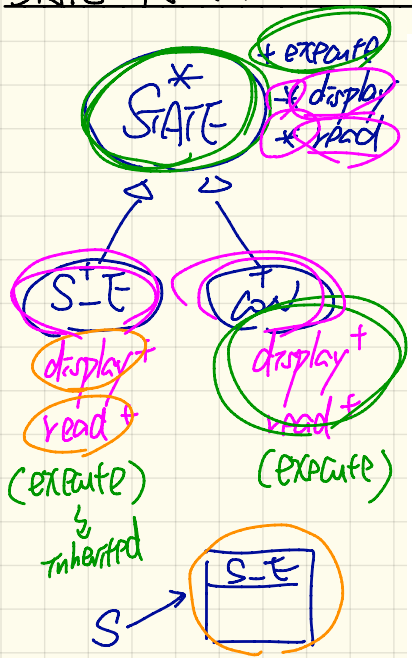
STATE PATTERN : Architecture






```
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
```

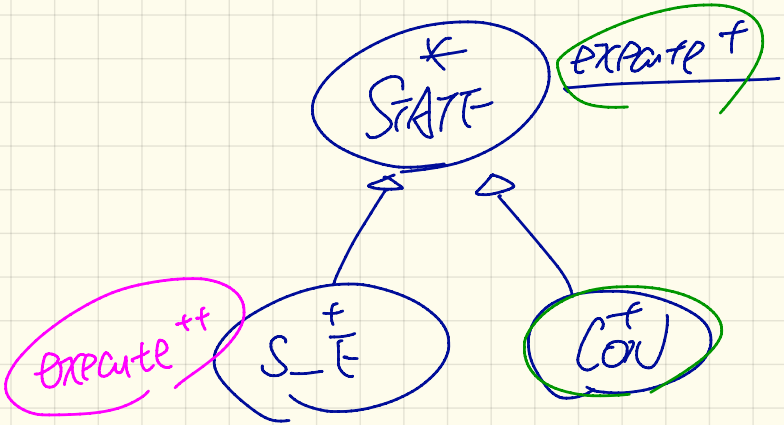
```
execute
  local
    good: BOOLEAN
  do
    from
    until
      good
    loop
      display
      -- Get answer and choice
      read
      good := correct
      if not good then
        message
      end
    end
    process
  end
end
```

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

version in STATE

pattern of calling helper features

TEMPLATE



S: STATE

Create {S_E} S.make

S.execute ← execute

Create {CON} S.make

S.execute ← execute

STATE PATTERN: Application Module

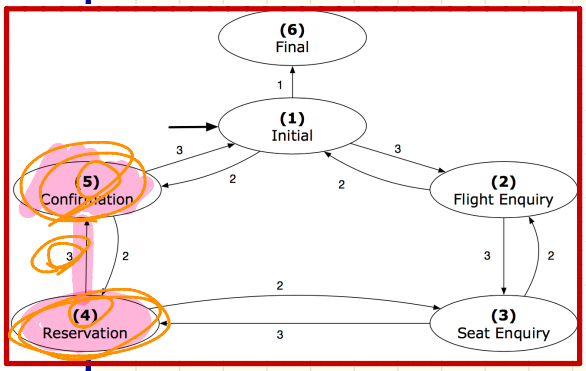
```
class APPLICATION create make
feature {NONE} -- Implementation of Transition Graph
  transition: ARRAY2[INTEGER]
    -- State transitions: transition[state, choice]
  states: ARRAY[STATE]
    -- State for each index, constrained by size of 'transition'
feature
  initial: INTEGER
  number_of_states: INTEGER
  number_of_choices: INTEGER
  make(n, m: INTEGER)
    do number_of_states := n
       number_of_choices := m
       create transition.make_filled(0, n, m)
       create states.make_empty
    end
feature
  put_state(s: STATE; index: INTEGER)
    require 1 ≤ index ≤ number_of_states
    do states.force(s, index) end
  choose_initial(index: INTEGER)
    require 1 ≤ index ≤ number_of_states
    do initial := index end
  put_transition(tar, src, choice: INTEGER)
    require
      1 ≤ src ≤ number_of_states
      1 ≤ tar ≤ number_of_states
      1 ≤ choice ≤ number_of_choices
    do
      transition.put(tar, src, choice)
    end
```

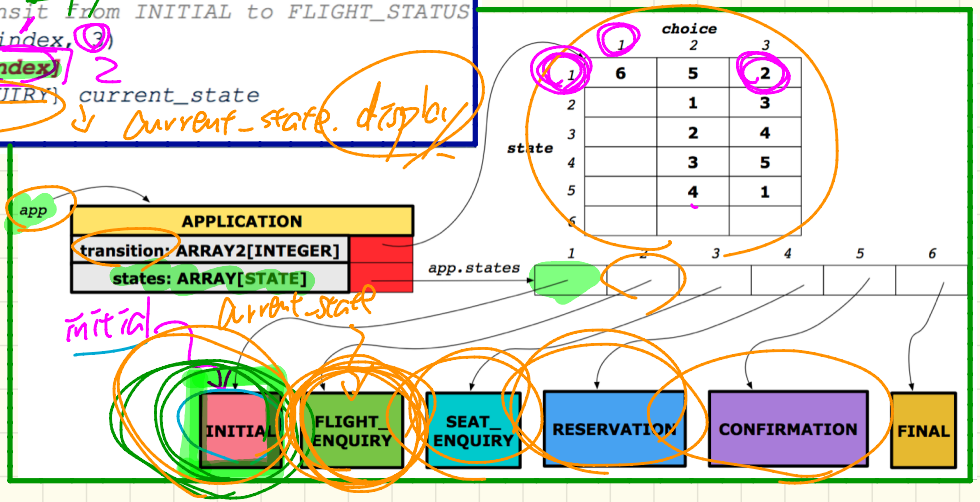
```
invariant
  transition.height = number_of_states
  transition.width = number_of_choices
```

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
```

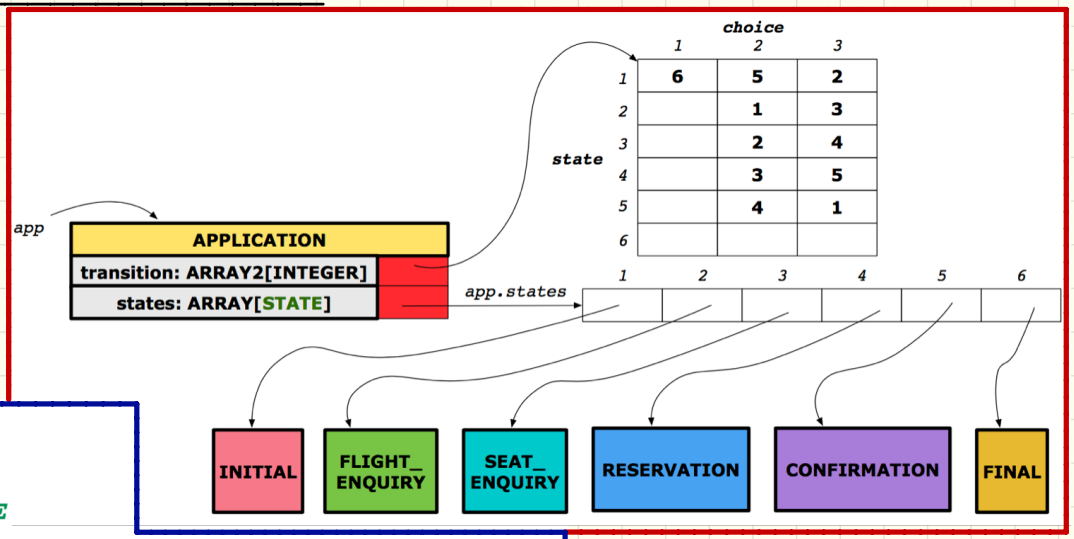




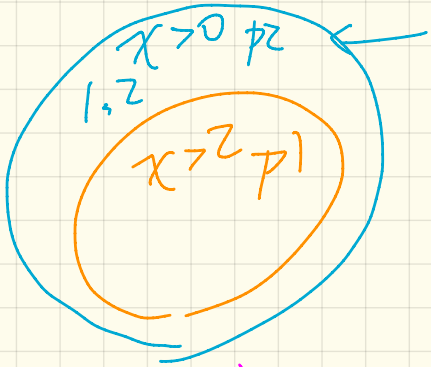
app.put_trans(6, 1, 1)
src act

app.put_trans(5, 4, 3)

State Pattern : Interactive Session




```
feature
  execute_session
    local
      current_state: STATE
      index: INTEGER
    do
      from
        index := initial
      until
        is_final (index)
      loop
        current_state := states[index]  -- polymorphism
        current_state.execute  -- dynamic binding
        index := transition.item (index, current_state.choice)
      end
    end
end
```



Sqrt (x: Int)

P1: x > 2 → 3, ...

P2: x > 0 → 1, 2, 3, ...

- P2 require less than P1

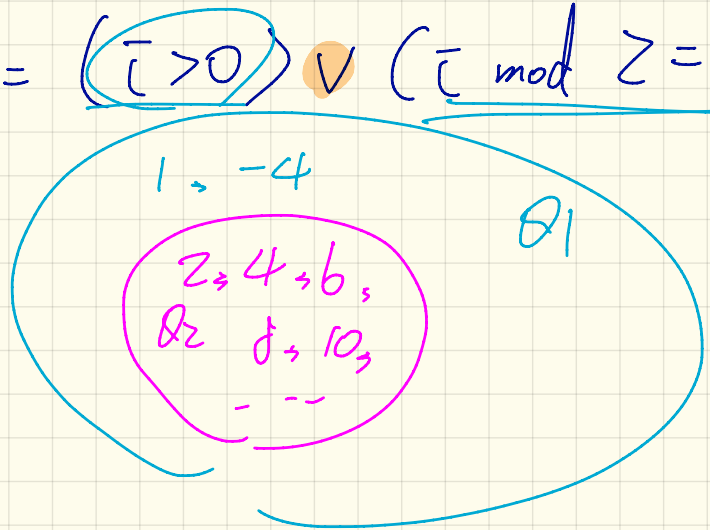
= P1 vs. P2 which one is correct?
It's up to your design decision.

Q_2: Result = $(i > 0) \wedge (i \bmod 2 = 0)$

Q_1: Result = $(i > 0) \vee (i \bmod 2 = 0)$

(1) $Q_2 \Rightarrow Q_1$

(2) $Q_1 \Rightarrow Q_2$



Subcontracting : Architectural View