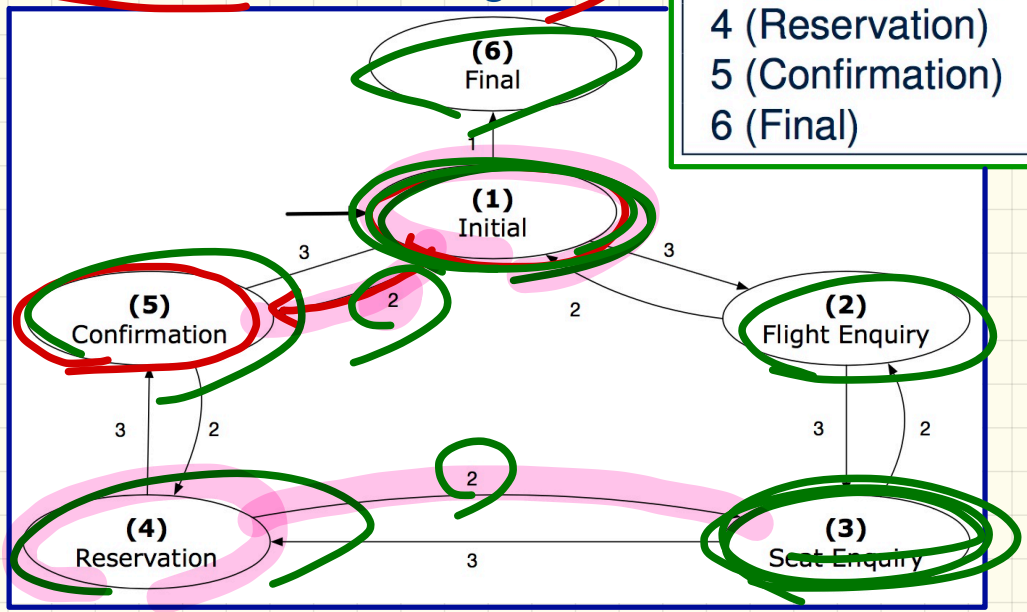
EECS 3311
Winter 2020
Monday March 9

Finite State Machine (FSM)

State Transition Table

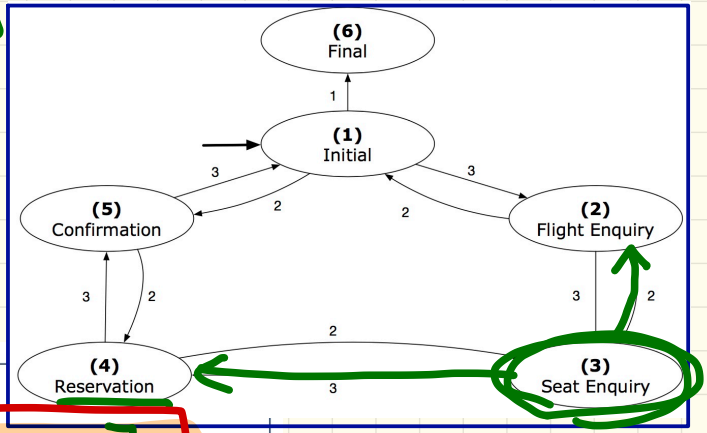
SRC STATE \ CHOICE	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

State Transition Diagram



Design of a Reservation System: First Attempt

Single Choice Principle
↳ when making a change
minimum # of places
to make such change

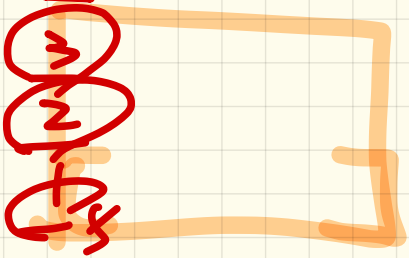



```
1_Initial_panel:
  -- Actions for Label 1.
2_Flight_Enquiry_panel:
  -- Actions for Label 2.
3_Seat_Enquiry_panel:
  -- Actions for Label 3.
4_Reservation_panel:
  -- Actions for Label 4.
5_Confirmation_panel:
  -- Actions for Label 5.
6_Final_panel:
  -- Actions for Label 6.
```

```
3_Seat_Enquiry_panel:
from
  Display Seat Enquiry Panel
until
  not (                    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
```

true ↳ exit

4_Reservation



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
```

Examples:

transition(3, 2)

transition(3, 3)

State Transition Table

APPROACH 2

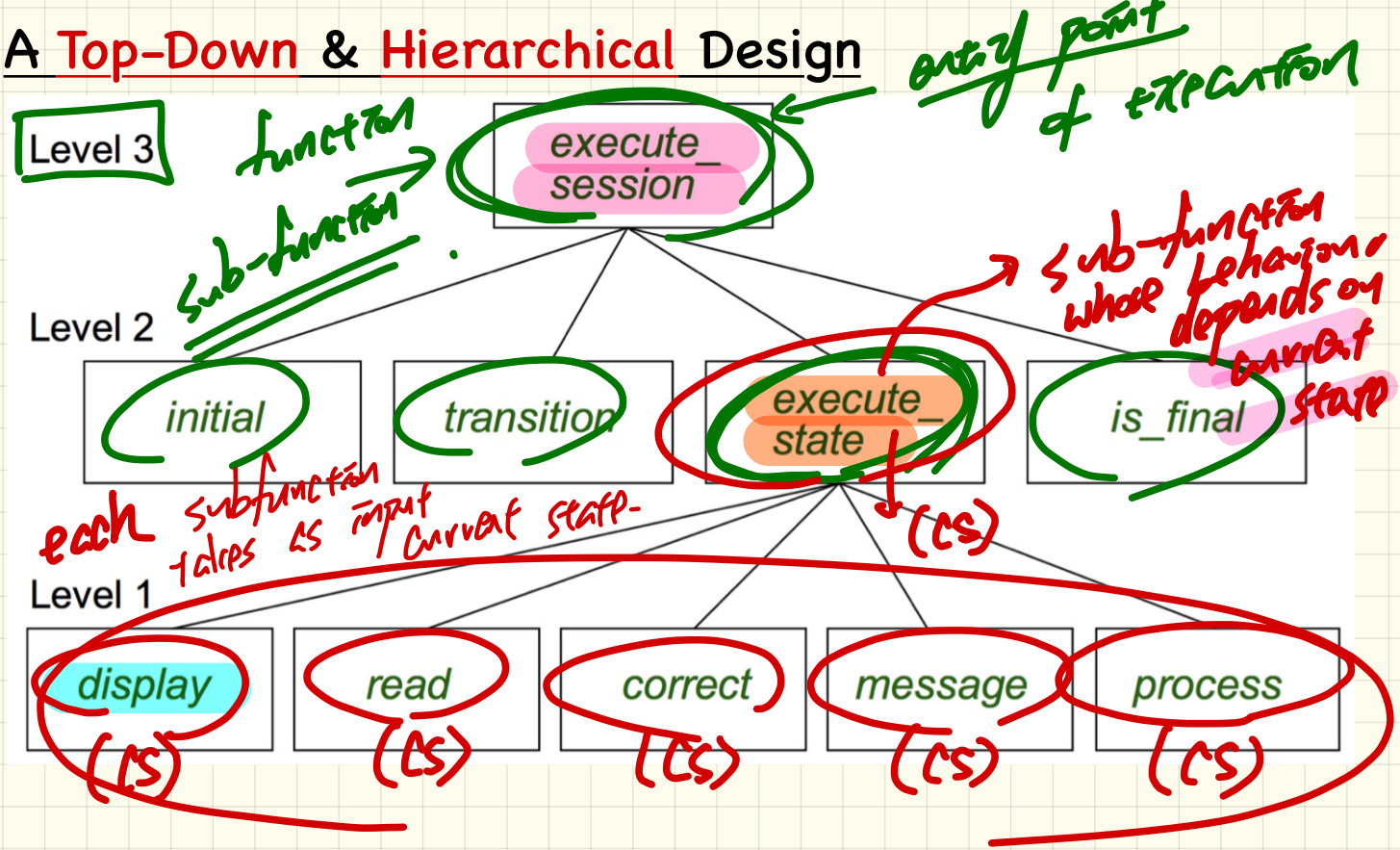
SRC STATE \ CHOICE	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

2D Array Implementation

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

Design of a Reservation System: Second Attempt (2)

A Top-Down & Hierarchical Design



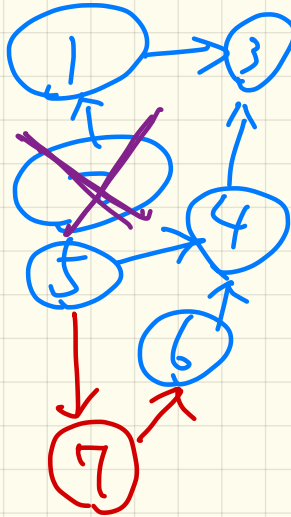
SCP

```
display( CS: INT )
do
  if CS = 1 then
  [
  elseif CS = 2 then
  [
  :
  elseif CS = 6 then
  [
end
  elseif CS = 7 then
  [
```

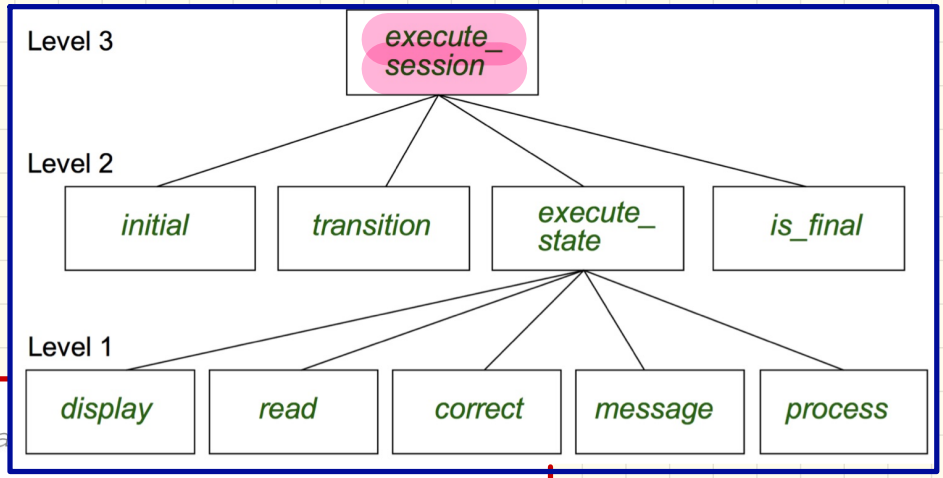
```
message( CS: INT )
do
  if CS = 1 then
  [
  elseif CS = 2 then
  [
  :
  elseif CS = 6 then
  [
end
  elseif CS = 7 then
  [
```

delete a state

add a new state



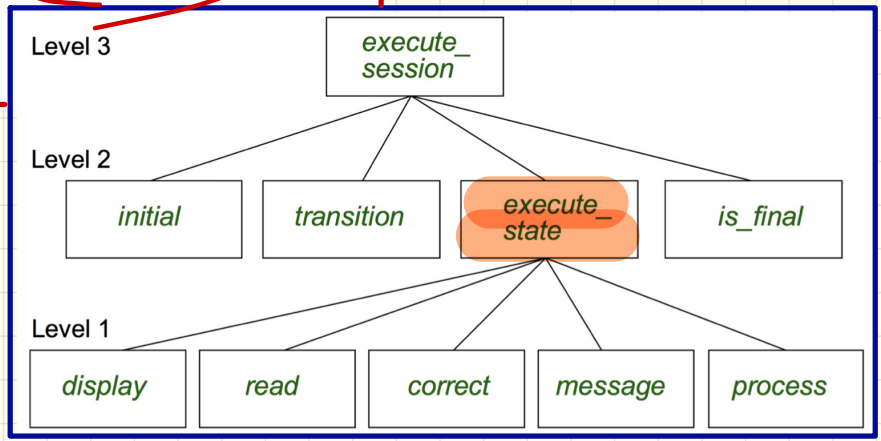
Design of a Reservation System: Second Attempt (3)




```
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
```

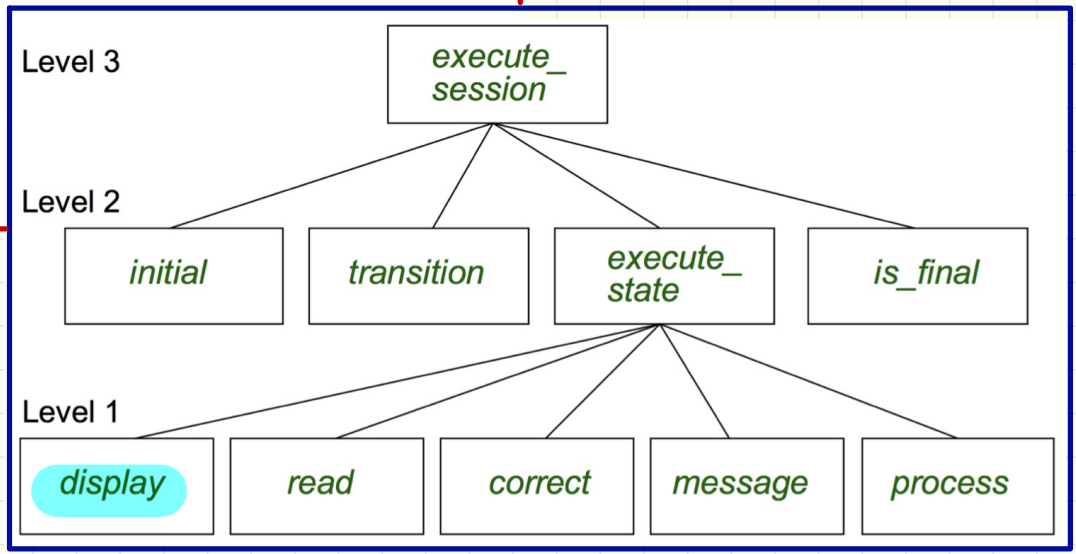
Design of a Reservation System: Second Attempt (4)

```
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
```



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
```



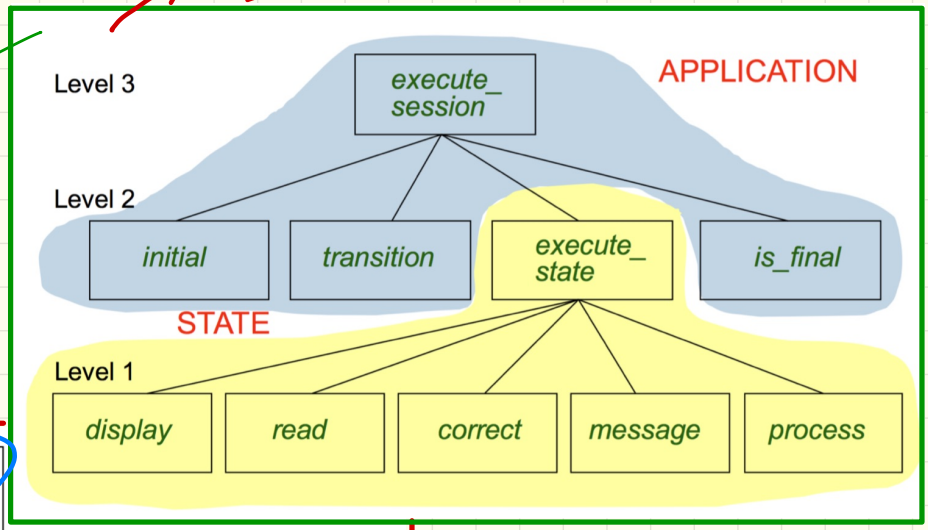
Moving from Top-Down Design to OO Design

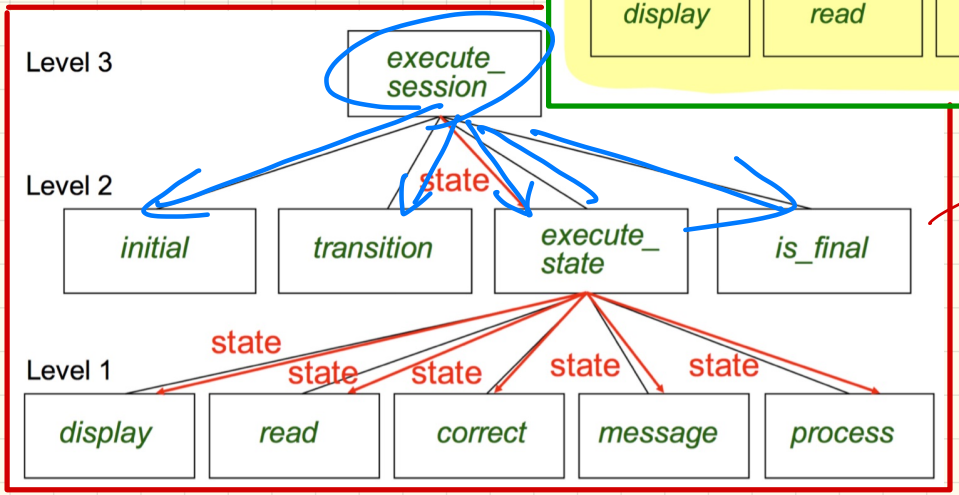
Polymorphism / Dynamic Binding

Object-Oriented

current_state: STATE
current_state.execute_session

current state is implicit.





Top-Down

current_state: INTEGER
execute_session(current_stste)

explicit input

multi shapes

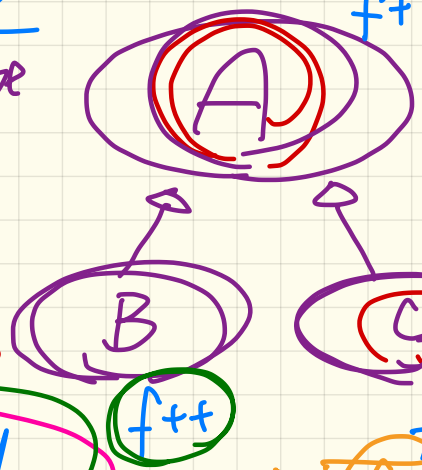
Polymorphism

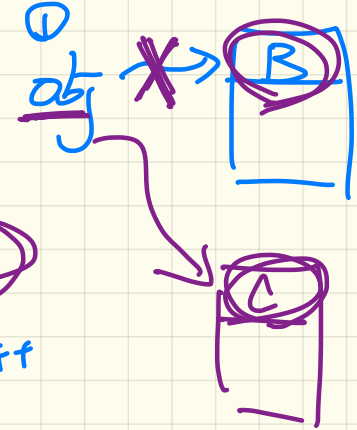
A obj = new B(..);

obj : A → static type

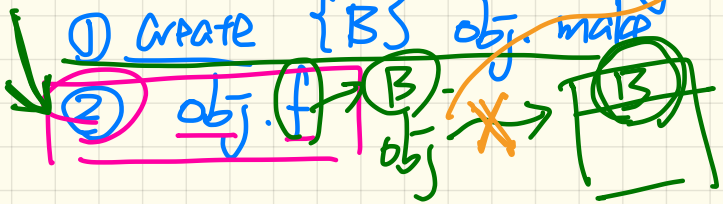
① Create {B} obj.make

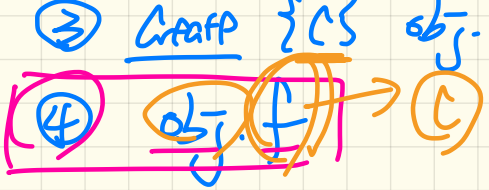
② Create {C} obj.make



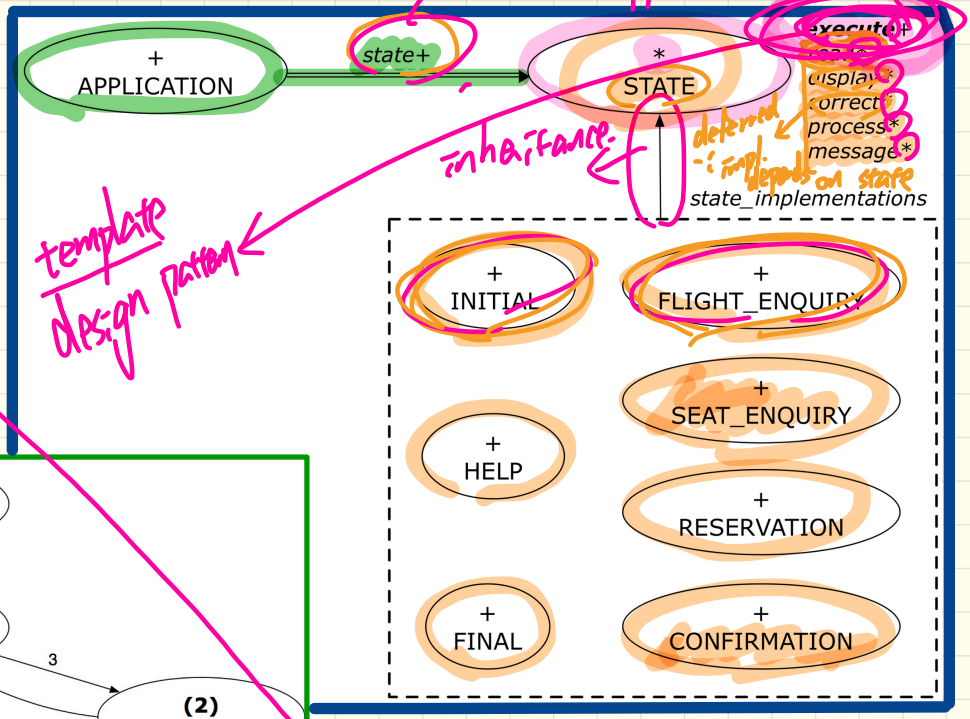


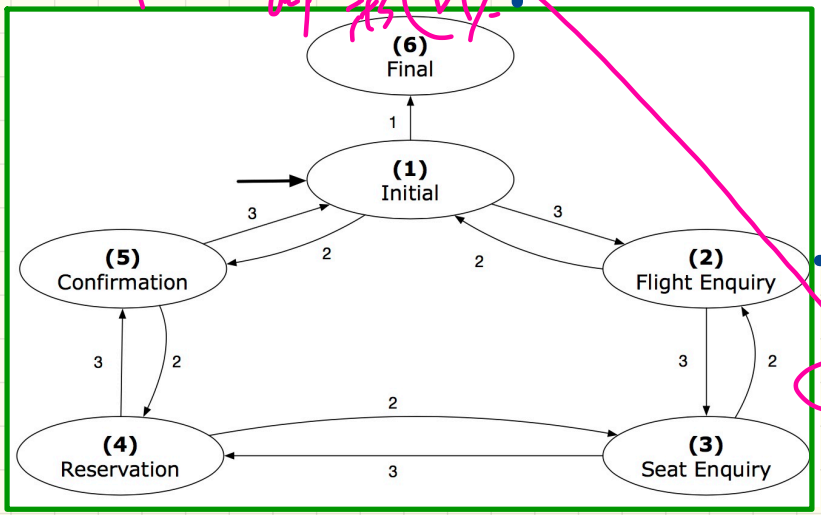
Dynamic Binding



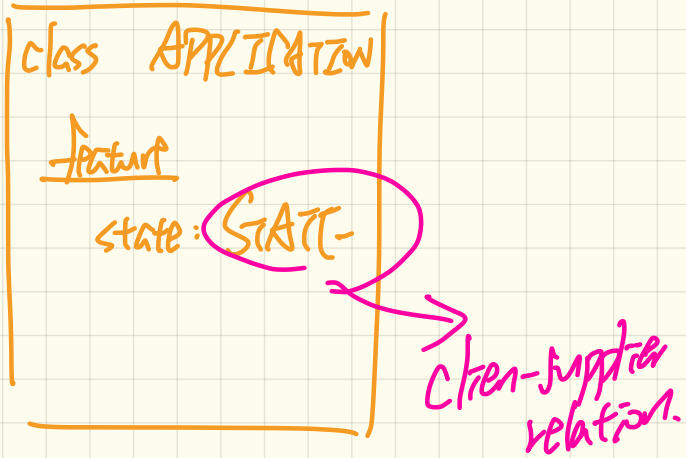


State Pattern: Architecture

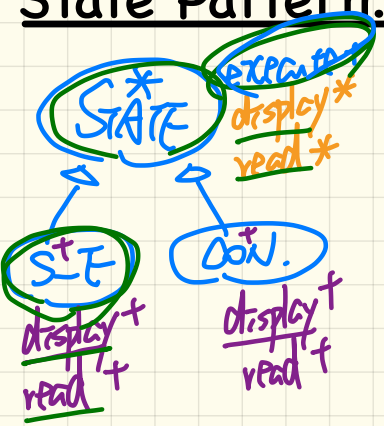


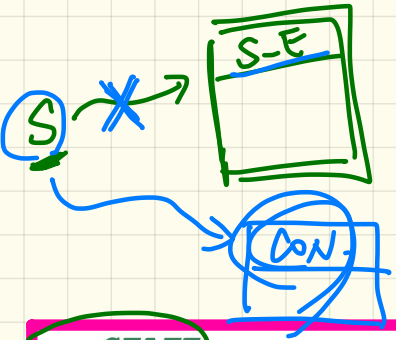



```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```


class APPLICATION
feature
state : STATE-
client-supplier
relation.

State Pattern: State Module






```
deferred class STATE
feature
  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
```

effective

```
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
```

TEMPLATE

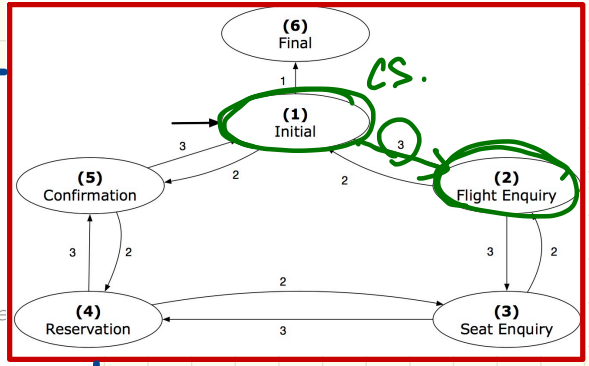
```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

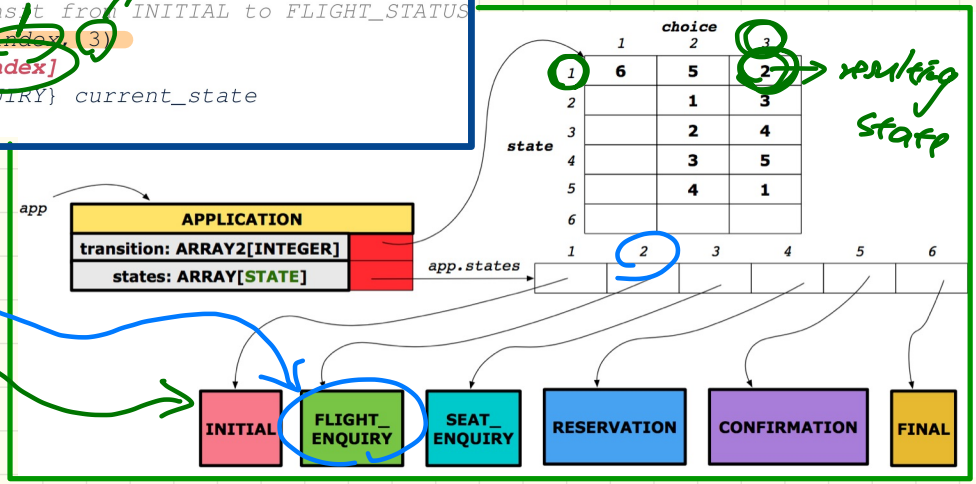
1. Version of execute to be called → Version in STATE
2. Version of display & read → S-E CON.

State Pattern: Test

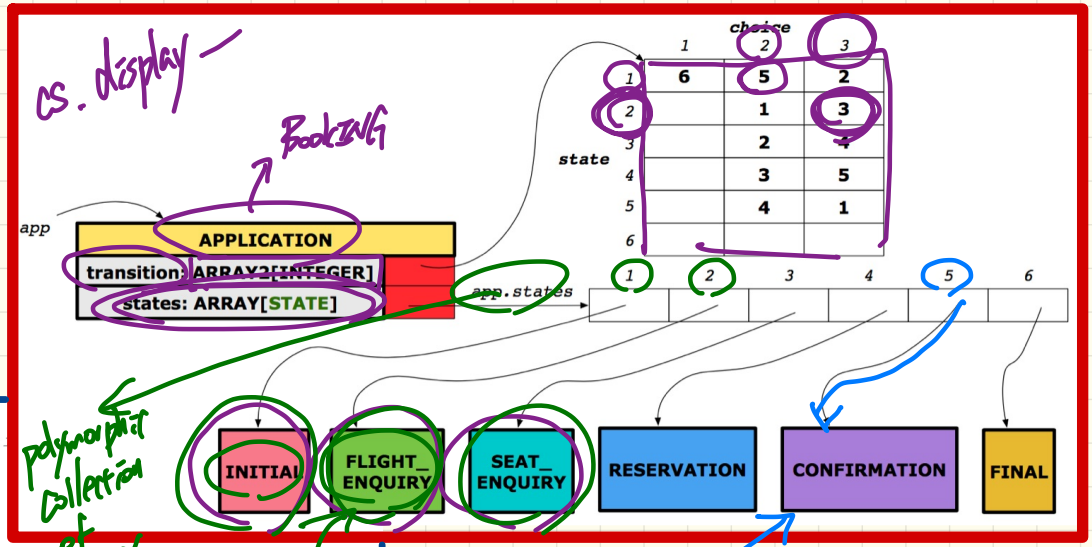
```
test_application: BOOLEAN
  local
    app: APPLICATION ; current_state: STATE ; index: INTEGER
  do
    create app.make (6, 3)
    app.put_state (create {INITIAL}.make, 1)
    -- Similarly for other 5 states.
    app.choose_initial (1)
    -- Transit to FINAL given current state INITIAL and choice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





State Pattern: Interactive Session




```
class APPLICATION
feature {NONE} -- Implementati
  transition: ARRAY2[INTEGER]
  states: ARRAY[STATE]
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
```

states : ARRAY[STATE]

cs : STATE

cs := states[2]

cs.display → v. F_E

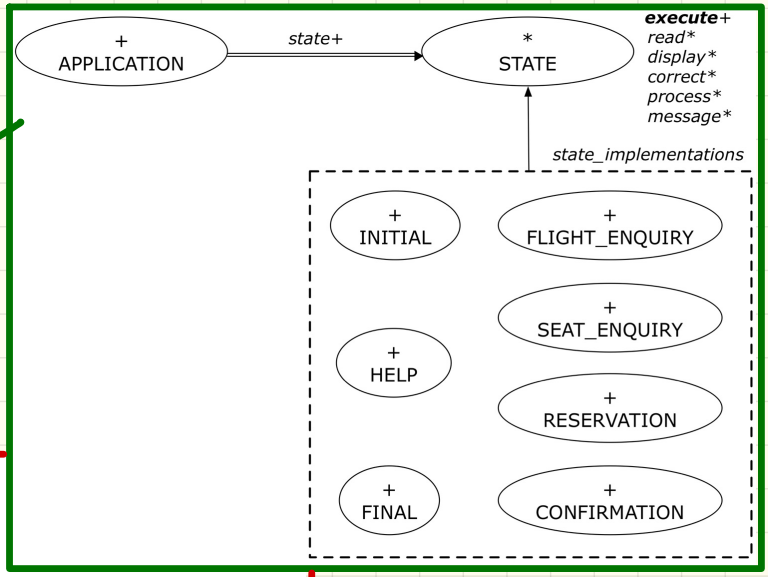
cs := states[5]

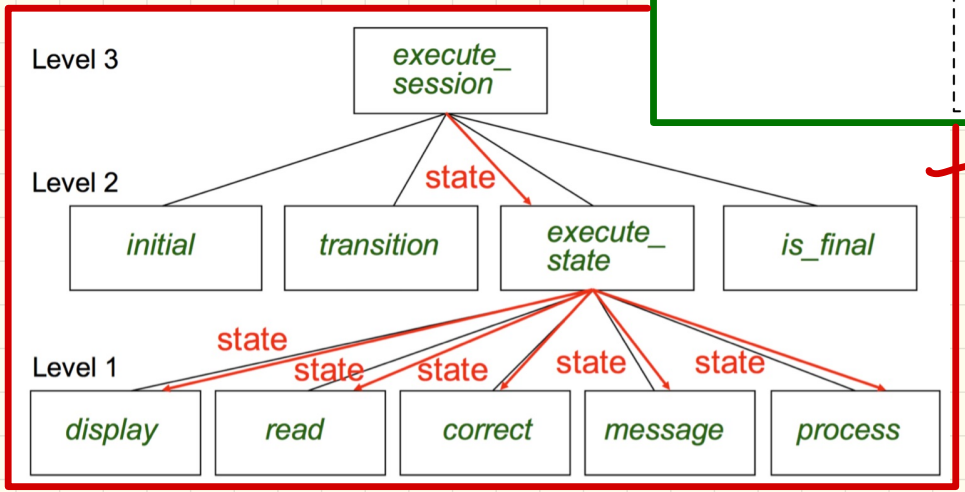
cs.display → v. Con.

Interactive System: Top-Down Design vs. OO Design

Object-Oriented

current_state: **STATE**
current_state.*execute_session*





Top-Down

current_state: **INTEGER**
execute_session(current_stste)