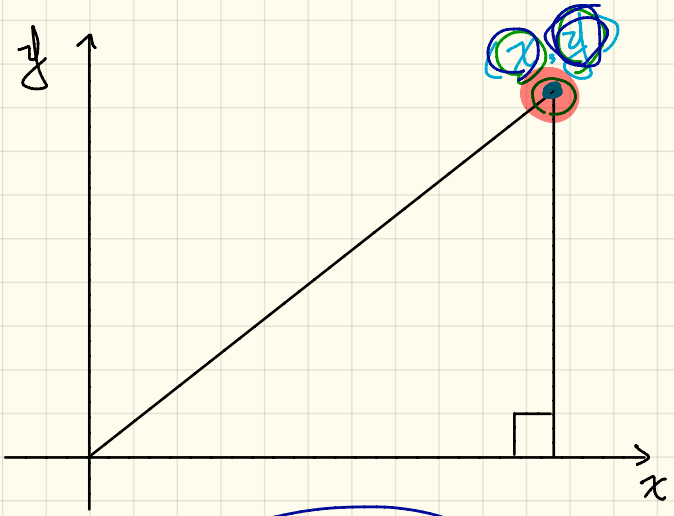
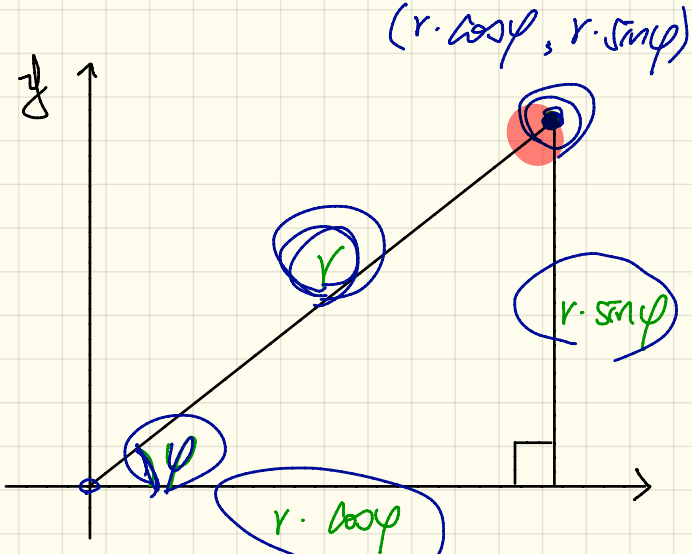
Tuesday Oct. 22

Lecture 12

Uniform Access to a 2D Point



Cartesian



Polar

Two Possible Ways to Implementing POINT

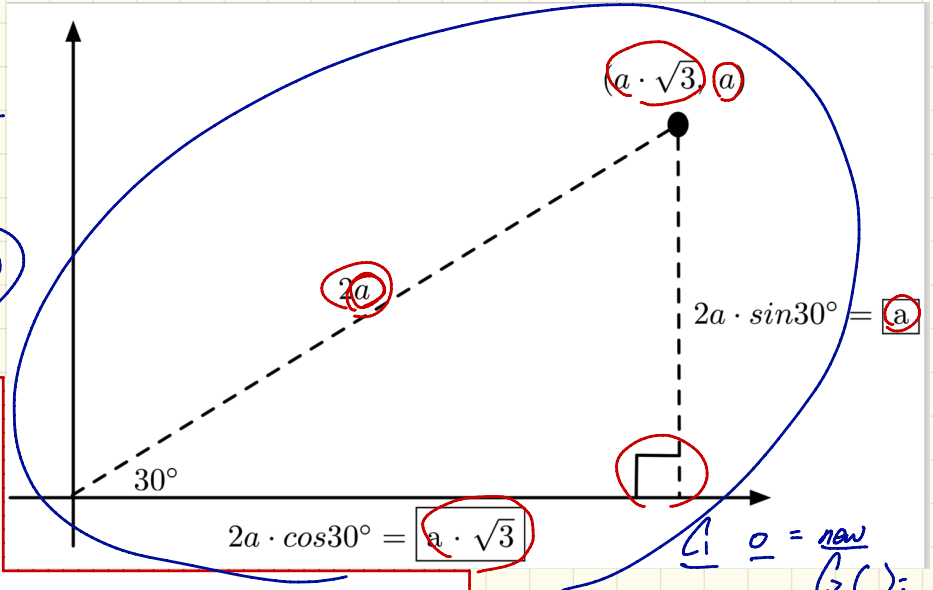
```
class POINT -- Version 1
feature -- Attributes
  x : REAL
  y : REAL
feature -- Constructors
  make_cartisian(nx: REAL; ny: REAL)
    do
      x := nx
      y := ny
    end
end
```

```
class POINT -- Version 2
feature {NONE} -- Attributes
  r : REAL
  p : REAL
feature -- Constructors
  make_polar(nr: REAL; np: REAL)
    do
      r := nr
      p := np
    end
feature -- Queries
  x : REAL do Result := r × cos(p) end
  x : REAL do Result := r × sin(p) end
end
```

Testing Uniform Access

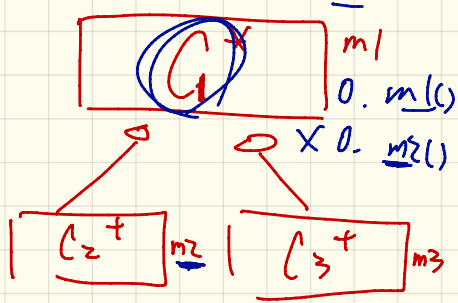
Point p1 = new Point()?

	Eiffel	Java
Attribute	o.f	o.a
Query	o.f	o.am()




```
test_points: BOOLEAN
  local
    A, X, Y: REAL
    p1, p2: POINT
  do
    comment("test: two systems of points")
    A := 5; X := A × √3; Y := A
    create {POINT} p1.make_cartisian (X, Y)
    create {POINT} p2.make_polar (2 × A, 1/6 π)
    Result := p1.x = p2.x and p1.y = p2.y
  end
```

C1 o = new C2();



Can Overloading support Uniform Access

YES

```
void ml (int i) {...} ←
void ml (String s) {...}
```

```
o. ml(2)←
o. ml ("alan")
```

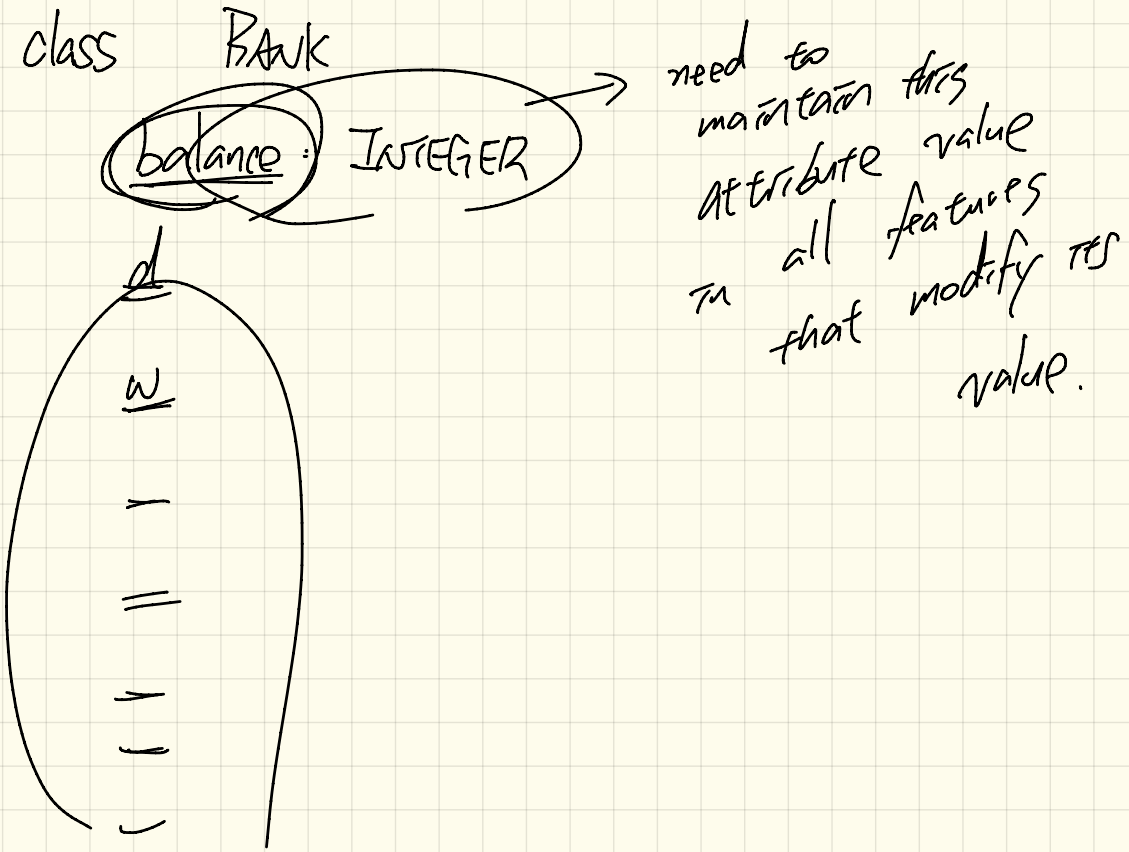
o. ml (23)

~~void ml (int i)..~~
~~void ml (int j)..~~

No

```
void m2 (int i)
void m2 (String s, int i)
```

o. m2 (2)
o. m2 ("a", 2)

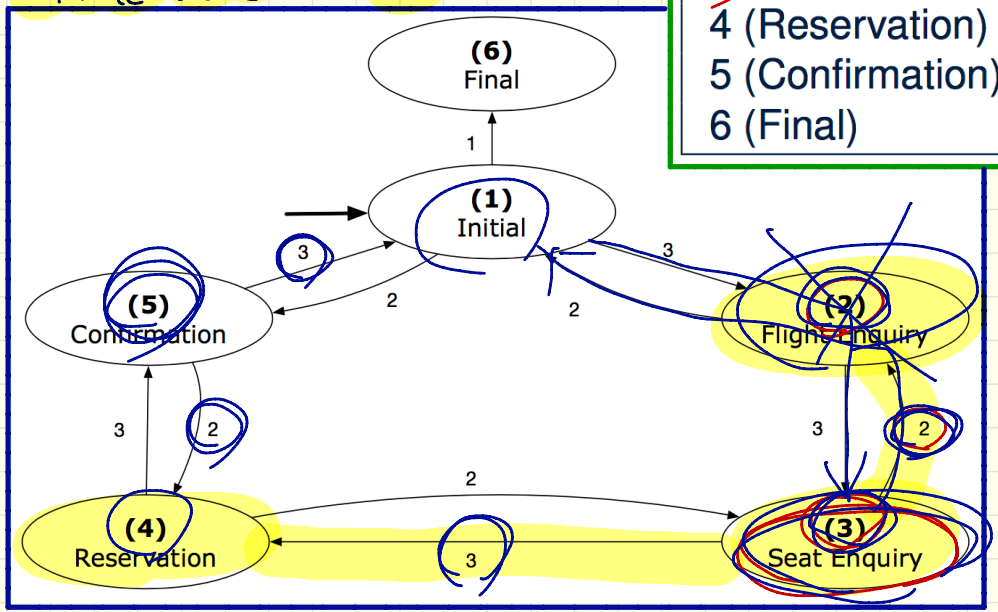

class BANK
balance : INTEGER
need to maintain this attribute value in all features that modify its value.
d
w

State Transition Diagram (FSM)

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)	–	–	–

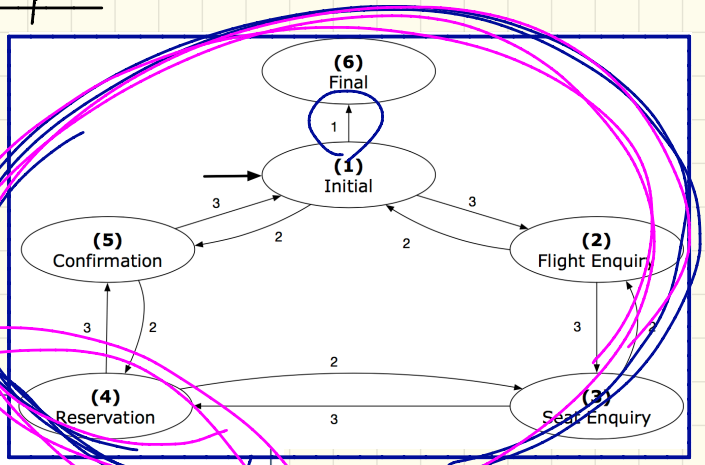
Finite State Machine



Design of a Reservation System: First Attempt

- code duplicates between labels

→ solution is not reusable for another problem




```
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
```

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
```

e.g. transition (3, 2) →
transition (3, 3)

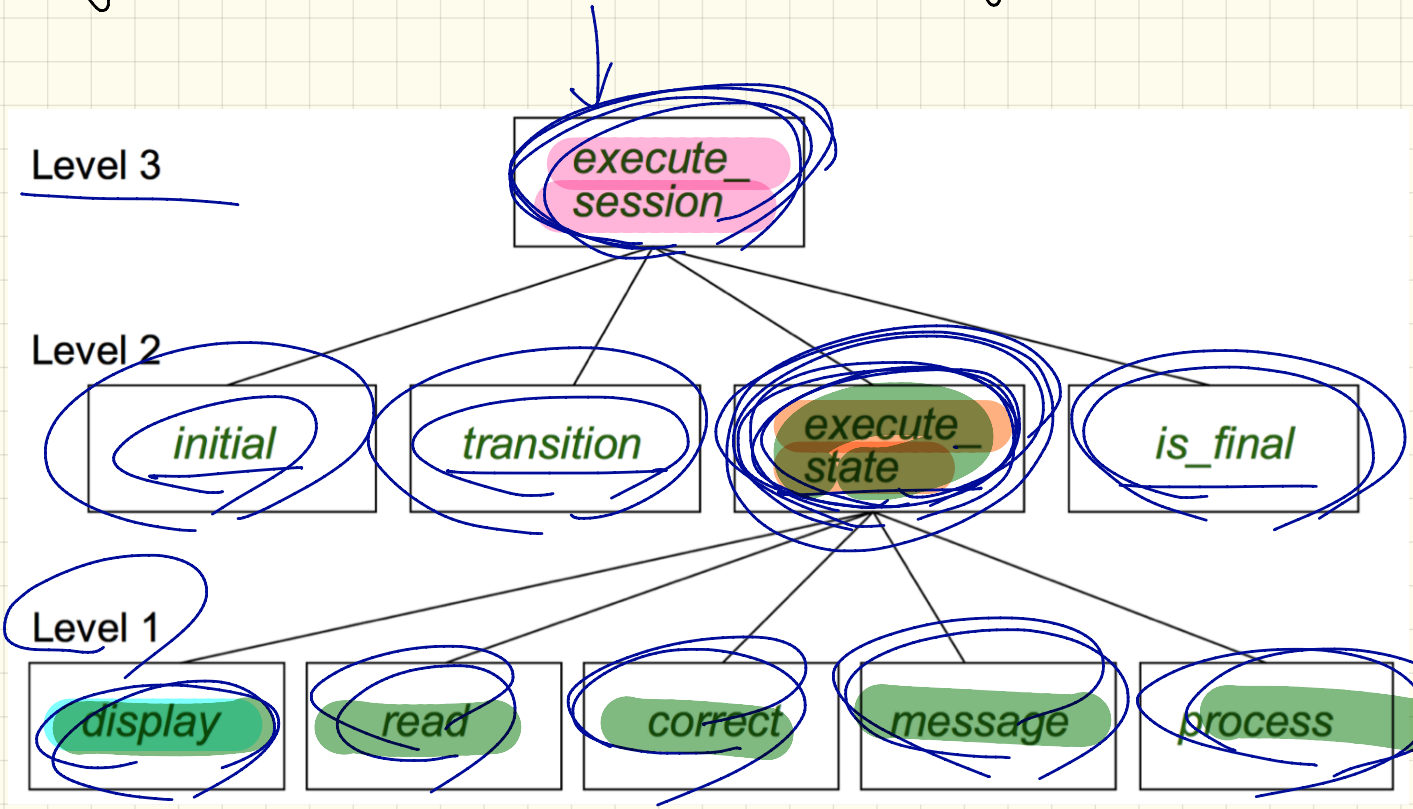
Transition Table

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)	–	–	–

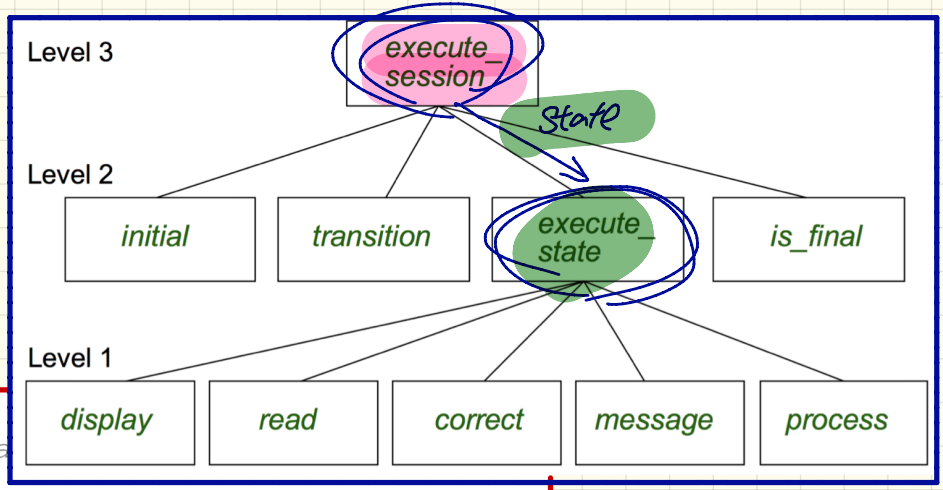
2D-Array Implementation

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

Design of a Reservation System: a Top-Down Design



Design of a Reservation System: Second Attempt (2)




```
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 (2)

```
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
```

display (current_state)

CS. display